Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

3D CNN Methods in Biomedical Image Segmentation

Supervisor:
Prof. Nico Lanconelli

Co-supervisor:
Prof. Renato Campanini

Co-supervisor:
Dr. Matteo Roffilli

Submitted by:
Filippo Maria Castelli

Academic Year 2018/2019

The real trouble with this world of ours is not that it is an unreasonable world, nor even that it is a reasonable one. The commonest kind of trouble is that it is nearly reasonable, but not quite... It looks just a little more mathematical and regular than it is; its exactitude is obvious, but its inexactitude is hidden; its wilderness lies in wait.

G.K. Chesterton

Abstract

A definite trend in Biomedical Imaging is the one towards the integration of increasingly complex interpretative layers to the pure data acquisition process. One of the most researched goals in the field is the automatic segmentation of objects of interest in extensive acquisition data, target that would allow Biomedical Imaging to look beyond its main use as a diagnostic-aid tool to become a cornerstone in ambitious large-scale challenges like the extensive quantitative study of the Human Brain. In 2019 Convolutional Neural Networks represent the state of the art in Biomedical Image segmentation and scientific interests from a variety of fields, spacing from automotive to natural resource exploration, converge to their development. While most of the applications of CNNs are focused on single-image segmentation, biomedical image data -being it MRI, CT-scans, Microscopy, etc- often benefits from three-dimensional volumetric expression. This work explores a reformulation of the CNN segmentation problem that is native to the 3D nature of the data, with particular interest to the applications to Fluorescence Microscopy volumetric data produced at the European Laboratories for Nonlinear Spectroscopy in the context of two different large international human brain study projects: the Human Brain Project and the White House BRAIN Initiative.

Sommario

Un chiaro trend nell'Imaging Biomedicale è quello che punta all'integrazione di strati interpretativi di crescente complessità nel processo di acquisizione del *dato puro*. Uno degli obiettivi più ricercati dell'ambito è la segmentazione automatica di oggetti di interesse in acquisizioni estese, traguardo che porterebbe l'Imaging Biomedicale a spingersi oltre il proprio utilizzo principale come strumento di ausilio diagnostico per diventare elemento fondante in challenge di larga scala quali l'analisi estensiva del cervello umano ad un livello quantitativo. Ad oggi (2019) le Reti Neurali Convolutionali (CNN) rappresentano lo stato nell'arte nella segmentazione di immagini biomediche ed il loro sviluppo è coadiuvato dalla convergenza di interessi di ricerca da svariati ambiti, dall'automotive alla ricerca di risorse naturali. Nonostante la maggior parte delle applicazioni delle CNN in tale senso sia focalizzata sulla segmentazione di singole immagini, il dato biomedicale -siano acquisizioni CT, MRI, microscopia, etc- molte volte beneficia di espressione volumetrica in tre dimensioni. Questo lavoro esplora la possibilità di riformulazione del problema di segmentazione con CNN in un formato nativo alla natura tridimensionale del dato, con particolare interesse per le applicazioni a dati di Microscopia a Fluorescenza prodotti ai Laboratori Europei di Spettroscopia Nonlineare (LENS) nel contesto di due iniziative internazionali nello studio del cervello umano: lo Human Brain Project e la White House BRAIN Initiative.

Acknowledgements

I would first like to thank my thesis advisor Prof. Nico Lanconelli of the Dept. of Physics and Astronomy at the University of Bologna and my thesis co-advisors, Prof. Renato Campanini and Dr. Matteo Roffilli, for their constant guidance, participation and support: their uninterrupted direction is what made this work possible and their navigation what helped me find my own way.

I would also like to acknowledge Dr. Ludovico Silvestri of the Dept. of Physics and Astronomy at the University of Florence and Biophotonics Group at the European Laboratories for Nonlinear Spectroscopy as I am gratefully indebted to his helpfulness during all the stages of this work.

I must express my very profound gratitude to my parents and to my family for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without their support. Thank you.

A last special thanks goes to Prof. and Turing Laureate Yann LeCun, VicePresident and Chief AI Scientist at Facebook for personally approving the use of his quotes in this work [Fig. 1].

Author
Filippo Maria Castelli

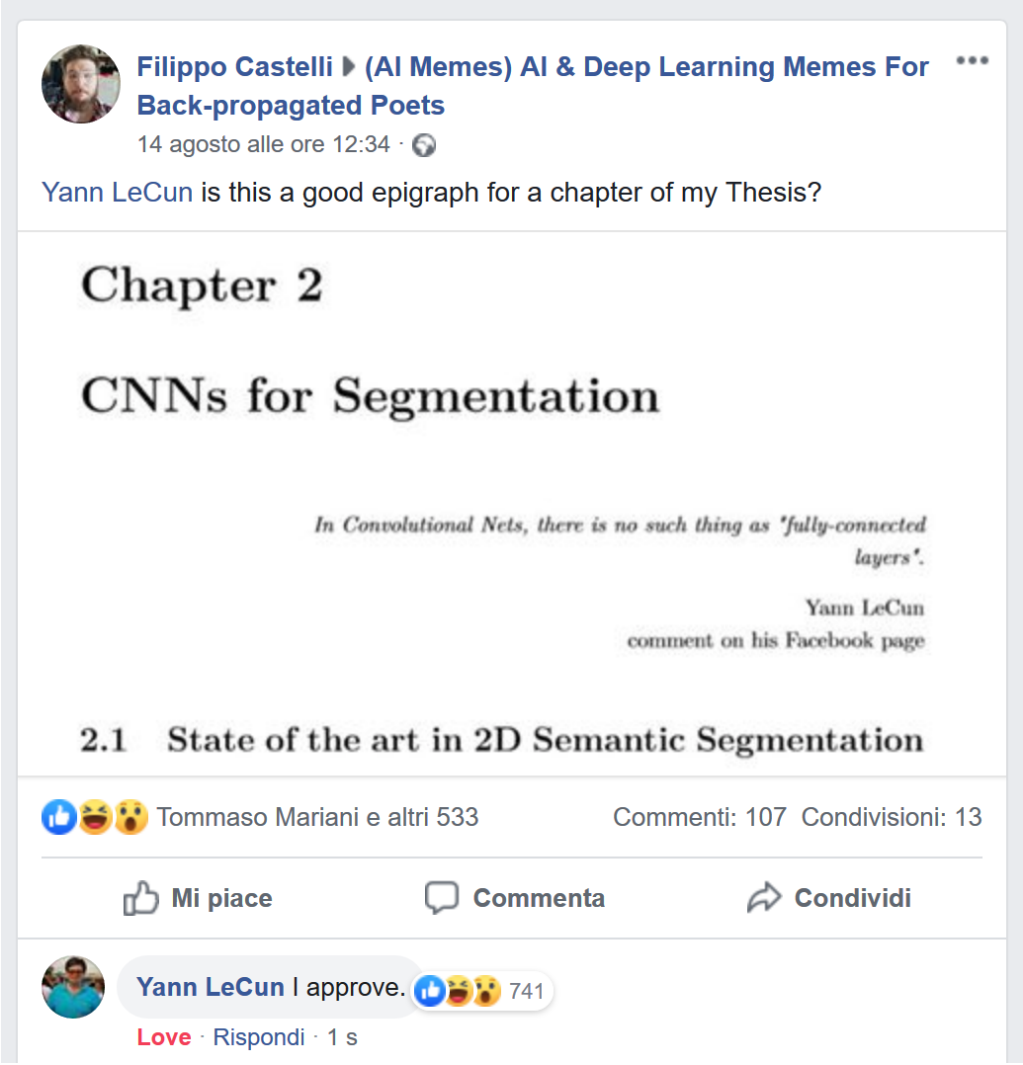

Figure 1: *"I approve."* - Yann LeCun

Contents

List of Figures

Part I

Introduction and State of the Art in Image Segmentation

Chapter 1

Deep Learning and Introductory Concepts

The paradigm for intelligence was logical reasoning, and the idea of what an internal representation would look like was it would be some kind of symbolic structure. That has completely changed with these big neural nets.

Geoffrey Hinton

1.1 Introduction

The almost totality of modern science and engineering has been built upon a plurimillenary intellectual framing based on *first principles* that pre-dates Aristotle: reality is something we, as humans, can aspire to understand logically and we do so by operating simplifications and generalizations to identify underlying "pure" rules and principles that describe *how* a system works. Using this approach, physical, biological and social systems are being thought and translated in mathematical forms to be then verified and finely tuned in their parameters using experimental data. This kind of reasoning is what we, as humans, perceive as the most effective path to describe reality in a reproducible way and this intuition is validated by the most evident of facts: it has always worked until now. Knowing this it's trivial to observe that this kind of setting naturally shaped the entirety of our knowledge, societies

and technological advancements, simply because there was no other logical road to follow.

Problems with this whole meta-epistemological setting arise when we face the study of systems that are too complex to allow mathematical descriptions: in absence of *first-principles* (in a loose sense) we have to resort to approaches that don't necessarily rely on our ability to build logical and mathematical models. In the last 40 years the large availability of digital technology and the drop of sensor prices have made possible a paradigm shift in scientific knowledge development and, consequentially, in the way we create models of reality: we've passed from classical modeling based on first principles to using large quantities of data to infer models, from using data to validate, verify and calibrate models to making data itself the focus and the first step of model creation.

Inspecting the matter from a slightly different perspective, from the front of technological development, we could affirm without any reasonable doubt that computers and computing represent its highest and it shouldn't come as a surprise that they implicitly reflect the way of thinking they were conceived into: the way computers classically solve tasks is to reproduce sequences of commands wisely put together by a programmer, a human who has previously thought the task itself, has it in his mind with perfect mathematical clarity and has deployed in machine-readable language the instruments to automate his mental model.
Or at least it used to be this way.
The parallel process to the scientific modeling paradigm shift in the computational field - or just another perspective of the same course - is the transition from programming *first principles* in the form of definite instructions ordered to the solution of a task, to the use of data to make the system *learn* in autonomy how to solve said task: the scientific complex responsible for the development of statistical algorithms, tools and knowledge to make it possible for a machine to *learn* how to solve tasks is complexively known as Machine Learning.

1.1.1 Historical Notes: ANNs and Deep Learning

In this work we are going to focus on a particular subset of this paradigm-shifting study field called *Deep Learning* (DL).

DL makes use of Artificial Neural Networks (ANNs) - models vaguely inspired by the biological neurons of the human visual cortex - to create algorithms that can *learn* how to solve a task from data.

Theoretical foundations for ANNs can be traced back to the early 40s with the work of McCulloch on neural models [82], the first learning hypothesis for such machines were made by Hebb in the late 40s [46], leading, in the late fifties, to the first true classifier based on such premises: Rosenblatt's *Perceptron* [100].

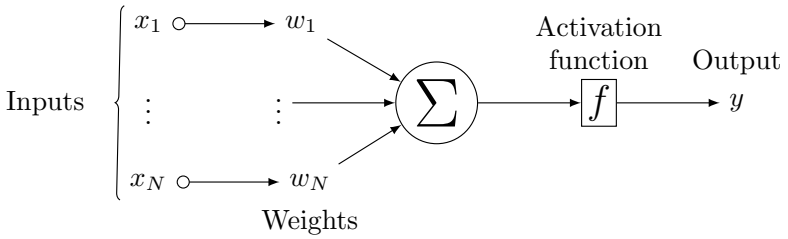

Figure 1.1: Rosenblatt's Perceptron Functional Scheme

The idea behind the Perceptron was simple but effective and is schematically represented in figure 1.1: as a biological neuron receive inputs from synapses and propagates conditional output along its axon, the Perceptron as a number of inputs and produces responses as linear combinations of those. The linear combination's weights determine relative importance of inputs and, by adjusting them to specific values, one could make a simple classifier. The right set of weights could be searched using a feedback system that iteratively compared expected results with actual ones and updated the vector formed by the weights proportionally to the difference between the two: the result was a set of weights that proved to be able to solve simple classification tasks.

The Perceptron was a very attractive solution for classification problems and was expected scale extremely well but it suffered from a deep-rooted issue: in 1969 Minsky proved that single layer Perceptrons couldn't handle any non-separable problem [86], at least not with the set of available knowledge at the time. This was particularly dramatic because, while a single Perceptron (or equivalently a layer of Perceptrons, for multiple output categories) was able to solve many classification tasks, if linear activation functions are used, linearity properties made stacking multiple layers of

this kind, one connected to the previous [1],formally equivalent to using a single layer: this property was convenient considering the lack of a working multi-layer training strategy but, on the other side, implied that XOR problems [2] could be solved only using multilayer architectures with nonlinear activation functions that couldn't be easily trained.
Theoretical issues with the Perceptron's ability to resolve XOR problems lead to an extensive lose of interest in ANNs from the scientific community, in the event that's known as the *First AI Winter*. This state of things lasted until the mid-seventies, when Werbos's *backpropagation* algorithm [125], based on a surprisingly simple application of the derivation chain rule, made feasible efficient learning for multilayer neural networks. During the following fifteen years Artificial Neural Networks saw renewed enthusiasm thanks to the promises of unseen model plasticity that they brought: suddenly there was an efficient algorithm to train those models and preliminary results were raising expectations to unseen levels, even making their way into pop culture. Unfortunately, the research community's participation went cold again when, despite the optimistic scenarios supported by theoretical advancements in the field, it became clear that only expensive hardware, mostly created ad-hoc, could possibly run computations that, although being relatively simple, came in way larger numbers than those computers of the era could handle, with resulting commercial systems being extremely expensive to create and to purchase. This ultimately diverted remaining interest and sealed ANNs in what was historically known as the *Second AI Winter*.

Even modest sized nets required a really high number of parameters to be trained, and the only possible solution to the problem was some heavy form of parallelization. During the 1990s and during all the 2000s parallel computing became major a topic of interest, massive supercomputing facilities were being built to solve all kinds of tasks and the possibility to parallelize this kind of computations was taken again in consideration. It wasn't until the 2009-2011 period that the current conception of Deep Learning came to be, as the Google Brain team, under suggestion of Prof.

[1] In this configuration the middle layers conventionally assume the name of *hidden layers*.

[2] The problem of predicting the outputs of a logic XOR (exclusive-OR) gate between two binary inputs: its peculiarity is that there exist no linear function that can separate the two output classes, meaning that the only working solutions have to be nonlinear separation functions.

Andrew Ng - who is now a key figure in the Deep Learning scene - started using Nvidia GPUs for the training algorithms.

Graphical Processing Units are perfect to parallelize extremely large numbers of repetitive calculations, such as geometrical transformations for 3D model rendering, and, with the right amount of work could, they be used to run the backpropagation algorithm with a fraction of the core-time CPU based systems would need. Switching from CPU computation to GPUs was the crucial step in dramatically scaling ANNs due to the large availability and reasonable price of pieces of hardware that were mainly marketed for videogames. This, combined with ease of programming provided by new tools like Nvidia's CUDA, determined an unprecedented spread in Deep Learning research and usage that is not going to slow down in any foreseeable future.

Deep Learning, especially after the recent developments, has changed the way we look at new problems and challenges by opening possibilities that were hard to think just ten years ago. Over the last six years - since the iconic AlexNet was brought to existence - we've come to witness not only the emergence of new solving strategies in an infinite number of contexts, but we are starting to see what could be defined - with a small amount of audacity and optimism - as a paradigm shift in the Kuhnian sense: the focus has now effectively shifted from *"how to extract the most effective features"* to *"what is the best architecture to help the model learn for itself"*.

Deep Learning has seen a massive application in almost all fields of science and industry in a very small amount of time, with massive resources being invested and improvements coming out at incredible rates while meeting both academical and industrial interests and producing applications that range from event identification in Particle Physics to autonomous driving and industrial automation. The reason of this is to be searched in DL's ability to obtain results with relative ease when compared to traditional ML methodologies: where the success of a given technique was largely influenced by the ML scientist's expertise in manually engineering synthetic features of the data for the model to compare and learn from, DL makes in most cases this kind of work obsolete thanks to its ability to produce implicit feature representations that - almost unexplicably - largely surpass in efficiency human-crafted feature selection.

On the other hand this also means that, despite the latest incredible leaps forward in network visualization [16], high dimensional inner representations of concepts and decision criteria can be too complicated to be human-readable. This makes DL act,

in many cases, as a "black box" the exact inner workings of which we struggle to fully understand. Perhaps the most fascinating aspect of Deep Learning is the emergent complex behaviour from building blocks of inherent simplicity and how simple but correct intuitions on these large scale systems's inner workings often introduce game-changing advancements (a fascinating example of this, to say one, is Goodfellow's simple and elegant explanation of adversarial example generation using nothing more than some basic linear perturbative theory [42] [17], that in the blink of an eye overturned the general perception of how a CNN classifier works in the public perception).

In this section we are going to review some of the basic concepts upon which most of the DL scenario is built upon and we will introduce some of the language that will be used in the the next chapters, although some rudimentary prior knowledge in machine learning is assumed from the reader.

1.2 Learning Algorithms

To talk about Deep Learning applications we have first to talk about what the *learning* in *Deep Learning* refers to.
The first uses of the term *Machine Learning* can be traced back to the work of Arthur Samuel: a common definition of *Machine Learning* attributed to him[3] quotes [102]

Field of study that gives computers the ability to learn without being explicitly programmed.

This kind of definition doesn't fully answer *how* a machine learning algorithm learns or what learning actually means, on this subject a much more precise and insightful definition of what a *learning algorithm* is, is provided by Tom Mitchell [87]:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

[3]Although [102] has historically often been reported as source, the exact quote can't be found in the text but only close versions of it. Consistently with many previous textbooks and works I decided to leave a reference to the 1959 article.

A much more defined outline emerges here with three main components: a **set of tasks T**, **experience E** and some kind of **performance measure P**.

1.2.1 Learning Components: Experience

Machine Learning algorithms are genereally of two types: **unsupervised** or **supervised**, depending on what kind of "experience" is used. In the first case the algorithm has to find some kind of structural property in the given data and the final goal might be *clustering* (dividing the dataset in clusters of samples sharing some property) or *density estimation* of the probability function that generated the dataset: in these cases all the needed information is in the data itself and no extra explicit annotations on "what" the data actually represents is intrinsically needed.

In the latter case (and this case covers the vast majority of Deep Learning algorithms) each data point is coupled with a "label", i.e. a description of "what" the data represents: this extra information is needed if we wish to create an algorithm that implicitly associates some features of the data with some kind of classification, so that it can make predictions of class membership on new and unseen examples.

Specifically, in the case of Deep Learning, the experience **E** is usually provided in the form of a **dataset** i.e. a set of *examples* or *data points*, each characterized by a collection of measured *features*. Typically an *example* of it is a real vector $\mathbf{x} \in \mathbb{R}^n$ with each component x_i representing different feature: a 255×255 monochrome image could be interpreted as a 65025-dimensional real vector with each component associated to the intensity of the corresponding pixel [4].

1.2.2 Learning Components: Task

From the definition above we underline a simple concept: the set of tasks **T** is a distinct entity from the learning itself. This means that *learning* is just the means to attain ability to perform a specific task and not a task in itself. The variety of tasks Machine Learning (and Deep Learning) algorithms can absolve is particulary wide but can be subdivided into categories. Perhaps the most common task for a ML algorithm is *classification* of examples into a set of categories: given an example

[4] although it's much more common to represent data in a tensorial format: a 255×255 RGB image can easily be expressed as a $255 \times 255 \times 3$ tensor.

we wish to find the category it belongs to among a set of pre-defined categories. While other task can include *regression* of numeric input data, *probabilty density estimation*, *synthesis* of new data and many more, we are especially interested in *segmentation*. Segmentation generally refers to the task of performing pixel-wise classification in an image: the algorithm receives as input an image and outputs a map of class membership of every pixel of said image. Another step is represented by the simultaneous identification of different "instances" of a same object in the image, this kind of task is called *Instance Segmentation* while the other is known as *Semantic Segmentation*.

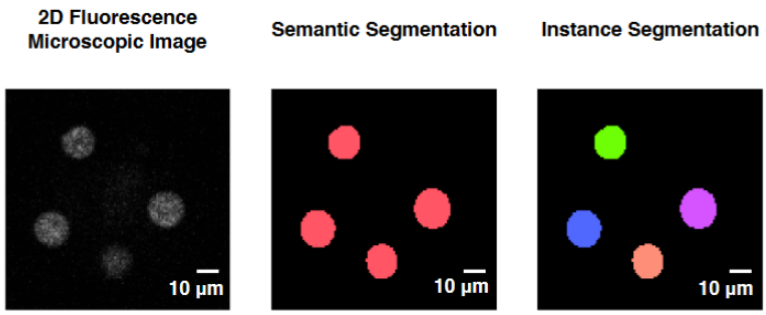

Figure 1.2: Semantic Segmentation vs Instance Segmentation [119]

The difference between the two might be clearer if we look at figure 1.2: in the case of *Semantic Segmentation* we are asking the algorithm to identify all the pixels belonging to the "cell" class, while in the case of *Instance Segmentation* we are not only asking the algorithm to identify "cell" pixels, but also to determine which pixels belong to each "cell" instance. In this work we are going to focus on 3D Semantic Segmentation although many of the used approaches can possibly be adapted to set up 3D Instance Segmentation problems.

1.2.3 Learning Components: Performance Measures

The performance measure **P** provides a quantitative estimate of *how well* the algorithm performs on the task **T**, and for that reason it has often to be tailored specifically for the current task. In classification tasks a good estimate of *how well* the model is performing can be an *accuracy* measure i.e. the proportion of examples for which the model produces the correct output. Performance measures can be

tricky to finely tune to the problem: if we wish to promote particular behaviors of the algorithm and we have no way of directly influence its decisions, then that kind of information must be encoded in the way the algorithm looks for an optimal solution. The typical example is represented by segmentation task performance evaluation in the case the background class is over-represented in the dataset and the foreground class is only present in a small portion of pixels: if we choose a performance metric that gives the same importance to background and foreground correct classification the algorithm would probably choose to trivially classify everything as background to improve its performance metric as it would be wrong in only a small percentage of the cases. To avoid this kind of problem metrics have to be chosen in such a way that they represent the desired behavior of the model. Furthermore, another typical restriction is the need for the performance metric to be a continuous and differentiable function, the sense of this request is to be comprehended in the next section.

1.3 Optimization, Loss, Gradient Descent

1.3.1 Gradient Descent Optimization

The previous section gives a very loose and generic description of how a learning algorithm should work: on a more practical basis we'd say that each learning algorithm is ultimately traceable to an optimization problem in which a *loss function*, giving an inverse performance on the current task (*how bad* the model performs on the task, typically the negative of the metric function from the above section) is minimized with respect to the parameters of the model itself.

More specifically, virtually all optimization algorithms for Deep Learning are based on Gradient Descent Optimization: Gradient Descent Optimization (GDO) is an algorithm used to minimize a loss function by iteratively moving the evaluation point in the direction of steepest descent, defined by the negative gradient of the loss function itself. The basic idea behind GDO is that, being $h(\mathbf{x})$ a function, defined and differentiable in the neighborhood of a point $\mathbf{x}_0$, for small enough $\epsilon \in \mathbb{R}^+$ the point

$$\mathbf{x}_1 = \mathbf{x}_0 - \epsilon \nabla_x h(\mathbf{x}) \tag{1.1}$$

is such that $h(x_1) < h(x_0)$. Applying this kind of reasoning in an iterative fashion

we can build a succession of points $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_N$ with the rule

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon \nabla_x h(\mathbf{x}) \tag{1.2}$$

such that the $h(\mathbf{x}_N) < h(\mathbf{x}_N - 1) < \cdots < h(\mathbf{x}_0)$ and such that the last member of the succession is a point arbitrarily close to a local minimum of $h(\mathbf{x})$ [5]. It's to be noted that the loss function, in order to be optimizable with gradient descent, should at least be differentiable, and this reduces the space of usable metrics.[6]

Adam Stochastic Gradient Descent

The basic formulation of 1.2 is just the simplest of many Gradient Descent algorithms. One of the most widely known Gradient Descent Algorithms is *Adam* (short for Adaptive Moment Estimation), introduced in 2014 by D. Kingma [60]. With respect to the "naive" GDO algorithms, Adam takes advantage of a few improvements: the learning rate is not fixed anymore but is adapted (hence the *adaptive* part of the name) for each of the model's weights using the first and second moments of the loss function's gradient. The loss function gradient, being evaluated over a random batch of data, can be considered to be a random variable. The first and second moments of the gradient are estimated using exponentially moving averages defined by

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w L_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_w L_t)^2 \end{aligned} \tag{1.3}$$

where L is the cost function, m_t and v_t are the estimators for the first and second moments of the gradient computed at the step t, while $\nabla_w L$ is the gradient on the current batch with respect to the weight parameter w.

To see that m and v are actually good estimators for the first and second moment of the gradient we compute their expectation values, anticipating to find something similar to

$$\begin{aligned} E[m_t] &\sim E[\nabla_w L_t] \\ E[v_t] &\sim E[(\nabla_w L_t)^2] \end{aligned} \tag{1.4}$$

[5] If the loss function is convex all local minima are also global minima and gradient descent converges to the global solution.

[6] For the avoidance of doubt, the differentiability is a requirement only to the loss function: other non-differentiable metrics can be used for keeping track of performance and fine tuning the model, it's just the training function that needs to be differentiable.

where equality would hold in the case we find m and v to be *unbiased estimators*.

Writing down the first terms of m

$$\begin{aligned} m_0 &= 0 \\ m_1 &= \beta_1 m_0 + (1-\beta_1)\nabla_w L_1 = (1-\beta_1)\nabla_w L_1 \\ m_2 &= \beta_1 m_1 + (1-\beta_1)\nabla_w L_2 = \beta_1(1-\beta_1)\nabla_w L_1 + (1-\beta_1)\nabla_w L_2 \\ m_3 &= \beta_1 m_2 + (1-\beta_1)\nabla_w L_3 = \beta_1^2(1-\beta_1)\nabla_w L_1 + \beta_1(1-\beta_1)\nabla_w L_2 + (1-\beta_1)\nabla_w L_3 \end{aligned} \tag{1.5}$$

we can see how we can express the moving average m_t in a much more concise way

$$m_t = (1-\beta_1)\sum_{i=0}^{t} \beta_1^{t-i}\nabla_w L_i \tag{1.6}$$

and computing its expected value

$$\begin{aligned} E[m_t] &= E[(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} g_i] \\ &= E[\nabla_w L_i](1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-1} + \zeta \\ &= E[\nabla_w L_i](1-\beta_1^t) + \zeta \end{aligned} \tag{1.7}$$

where in the second line we approximated $\nabla_w L_i$ with $\nabla_w L_t$, taking it out of the sum and introducing an error ζ that converges to zero, we then just treated the sum as a geometric series expansion. It's easy to see that for v holds the analologus

$$E[v_t] = E[g_i^2](1-\beta_2^t) + \zeta \tag{1.8}$$

The resulting estimators for $\nabla_w L$ and $\nabla_w L^2$ are *biased* so we have to introduce two correction terms:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{1.9}$$

The weight update for each individual parameter can then be expressed as

$$w_t = w_{t-1} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \tag{1.10}$$

where η is the learning step size and ϵ is a finite quantity for regularization, yielding a sequence of updates that is expected to converge faster than GDO.

1.3.2 Loss Functions

We introduced loss functions and said that they would need to be chosen in a problem-specific way, but how should I decide what loss function to adopt for my particular application? Maybe the question is more transparent if we look at it from a slightly different perspective by formalizing the learning problem in the classification context[7].

Let's say we have a space X of possible inputs and a space Y - typically $\{-1, +1\}$ in classification tasks - of target labels and we want to learn a function $f(\vec{x}) : \vec{x} \to \mathbb{R}$ that maps X to Y. Let's also assume that the data is actually generated by a joint distribution $p(\vec{x}, y)$ with $y \in Y$, which we do not know explicitly. The loss function we introduced above describes how different is the model's prediction from the actual "true value" and we indicate it by $h(f(\vec{x}), y)$ where $f(\vec{x})$ represents the prediction and y is the actual label. The goal of a learning algorithm would then be to minimize the expectation value of said function over the probability distribution of the data which can be simply expressed as the *risk integral* 1.11.

$$I[f] = \int_{X \times Y} h(f(\vec{x}), y) p(\vec{x}, y) d\vec{x} dy \tag{1.11}$$

The first thing to notice in the integral above is that it's not directly evaluable, mainly because we do not have access to the distribution $p(\vec{x}, y)$ (and trivially if we knew its values we wouldn't probably be searching for $f(\vec{x})$), instead we have access to a finite set of labeled couples $\{(\vec{x_1}, y_1), (\vec{x_2}, y_2), \ldots, (\vec{x_m}, y_m), \}$ that cannot possibly describe $p(\vec{x}, y)$ with full detail. What we do instead is approximating $p(\vec{x}, y)$ with an *empirical distribution*

$$p_{emp}(\vec{x}, y) \approx \frac{1}{m} \sum_{i=1}^{i=m} \delta(x_i) \delta(y_i) \tag{1.12}$$

where the index i counts all the available data points. What we obtain then is that, given that our dataset sufficiently describes $p(\vec{x}, y)$, we can get a good estimate of the risk integral 1.11 as

$$I[f] \approx \int_{X \times Y} h(f(\vec{x}), y) p_{emp}(\vec{x}, y) d\vec{x} dy = \frac{1}{m} \sum_{i=1}^{i=m} h(f(\vec{x}), y) \tag{1.13}$$

[7] These considerations are actually consistent with the segmentation cases as they can ultimately be seen as pixel-wise classifications.

Different types of loss functions help us encode different types of desired behavior, depending on their structure and mathematical formulation. A natural choice for a function that accounts for error in classification would trivially be

$$h(f(\vec{x}, y)) = H(-yf(\vec{x})) \tag{1.14}$$

where H is the Heavyside function: such loss function would be 1 when prediction and label share the same sign, and 0 otherwise. Perfectly simple at first glance, on a closer look one realizes that it doesn't meet the differentiability requirements and, because of this, the step function doesn't lend itself to an easy integration in gradient descent optimization algorithms. The next logical choice is to find other differentiable functions that express the same approximate behavior, here we take a look at some of the most common ones.

Square Loss

One of the simplest loss function should actually be quite familiar to anyone who has encontered least square regression at some point in their life. Square Loss is simply the squared difference between prediction and the actual label:

$$h(f(\vec{x}, y)) = (y - f(\vec{x}))^2 \tag{1.15}$$

Despite its extremely simple appearance, square loss provides a convex formulation and matches the 1.14 indicator function when $yf(\vec{x}) = 0$ and $yf(\vec{x}) = 1$. It can easily be seen that this kind of loss function heavily penalizes outliers in the dataset when $|yf(\vec{x})|$ takes high values: this is effect slows significantly convergence so it's not common to see square losses in practical applications.

Hinge Loss

Hinge Loss can be defined as

$$h(f(\vec{x}, y)) = \max(0, 1 - yf(\vec{x})) = |1 - yf(\vec{x})|_+ \tag{1.16}$$

Hinge loss matches 1.14 when $\text{sign}(f(\vec{x})) = y$ and $|yf(\vec{x})| \geq 1$ and doesn't penalize points with $yf(\vec{x}) > 1$. The 1.16 form, though, it's not differentiable in $yf(\vec{x}) = 1$ so, in this loss definition it's not suitable for gradient descent optimization: it's common to use instead a *Generalized Smooth Hinge Loss* defined as

$$h(f(\vec{x}, y))_\alpha = \begin{cases} \frac{\alpha}{\alpha+1} & \text{if } z < 0 \\ \frac{1}{\alpha+1} z^{\alpha+1} - z + \frac{\alpha}{\alpha+1} & \text{if } <= 0 < z < 1 \\ 0 & \text{if } z \geq 1 \end{cases} \tag{1.17}$$

where $z = yf(\vec{x})$ and α is a real parameter. This function provides the same general behavior as the Hinge Loss but in a differentiable format.

Cross Entropy Loss

Cross Entropy Loss is perhaps the most used Loss Function in Deep Learning, it's defined as:

$$h(f(\vec{x}, y)) = -t \ln \sigma(\vec{x}) - (1 - t) \ln (1 - \sigma(\vec{x})) \tag{1.18}$$

where $t = (1 + y)/2$ so that $t \in \{0, 1\}$ and we define $\sigma\vec{x}$ as

$$\sigma(\vec{x}) = \frac{1}{1 + e^{-f(\vec{x})}} \tag{1.19}$$

In this case there's no point that's assigned exactly zero penalty but strong predictions are encouraged. Class imbalance can be taken into account using weighted versions of this function, like the *Weighted Cross Entropy function (WCE)* defined as

$$h(f(\vec{x}, y)) = -\beta t \ln \sigma(\vec{x}) - (1 - t) \ln (1 - \sigma(\vec{x})) \tag{1.20}$$

where a real parameter β can be used as a to adjust for false positive or false negatives: $\beta > 1$ decreases the number of false negatives, $\beta < 1$ decreases the number of false positives

Another cross entropy variant is the *Balanced Cross Entropy function (BCE)*, it's defined similarly to 1.20 the parameter β this time controls also the negative examples:

$$h(f(\vec{x}, y)) = -\beta t \ln \sigma(\vec{x}) - (1 - \beta)(1 - t) \ln (1 - \sigma(\vec{x})) \tag{1.21}$$

1.3.3 Segmentation-Specific Metrics

The main question we want our performance metric to answer should be *"how well does the predicted set of pixels does approximate the ground-truth set?"*, it can be non-trivial to come with a definitive answer, especially when class imbalance and regularity requirements are taken into account. When evaluating the effectiveness of a segmentation task a few concepts can be borrowed from the Image Processing area: many metrics in Image Segmentation are based on general concepts of set similarity, with few additional class-balancing and regularization tricks.

Set Similarity Metrics

Most loss functions for Image Segmentation are based on one of two slightly different metrics: one is the Jaccard Similarity Coefficient and the other is the Dice Coefficient.

The Jaccard Similarity Coefficient is mainly known with the simpler and more intuitive name *Intersection over Union*. As the name suggest, Jaccard index is just a fancier name for the ratio of two terms: the first is the number of common pixels between the ground truth set and the predicted set, the second is the number pixels in the union of the two sets. Let's call our prediction and ground truth sets, respctively P and G:

$$IoU = \frac{|P| \cap |G|}{|P \cup G|} = \frac{|P| \cap |G|}{|P| + |G| - |P \cup G|} \tag{1.22}$$

Intersection Over Union is an extremely common metric for performance evaluation and hyperparameter selection but, in the simple formulation 1.22, can't be used as a loss function. On this front there are some recent attempts of approximation and generalization of Intersection Over Union metrics to loss functions, notably [97] and [128].

The other important metric for segmentation is the *Dice Similarity Coefficient*. The Dice Coefficient's formulation is very similar to Jaccard Coefficient's:

$$DSC = \frac{2|P \cap G|}{|P| + |G|} \tag{1.23}$$

Although being extremely similar to 1.22, when 1.23 is expressed in vector notation (expressing G and P as vectors) it becomes

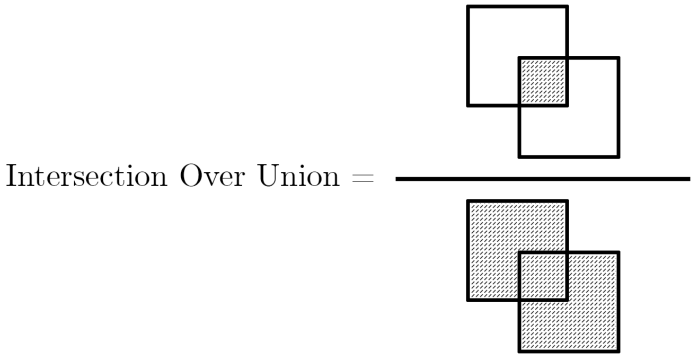

Figure 1.3: Intersection Over Union

$$DSC = \frac{2\,\langle \vec{p}|\vec{g}\rangle}{\|\vec{p}\|_2^2 + \|\vec{g}\|_2^2} = \frac{2\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} {p_i}^2 + \sum_{i=1}^{N} {g_i}^2} \tag{1.24}$$

where we used the ℓ_2 norm and N is total number of pixels (or voxels if we work in higher dimensionality spaces). Expression 1.24 can be differentiated w.r.t. an arbitrary pixel (voxel):

$$\frac{\partial DSC}{\partial p_j} = 2\frac{g_j(\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2) - 2p_j(\sum_{i=1}^{N} p_i g_i)}{(\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2)^2} \tag{1.25}$$

to be used as a proper loss function in gradient descent algorithms.

It's important to notice that the response of the model isn't directly a 0-1 classification but rather a probability map with real values between 0 and 1, both of these metric families become dependent on the particular choice of probability level over which we consider a pixel to be positively segmented.

1.4 Overfitting, Underfitting, Model Capacity

The optimization process has the goal of minimizing the loss function with respect to the model's parameters, but simple minimization of the loss function on a finite dataset, though, is not *per se* guaranteed to yield the best outcome. This is mainly ascribable to two order of reasons. When introducing loss functions we tried to minimize the risk integral 1.11 by approximating the data generator distribution with

an empirical one 1.12, we affirmed that this approximation should hold when the training dataset is *descriptive enough* of the real distribution. One could suppose that the level of descriptive accuracy should be high enough that we're able to accurately reproduce the general trends with an high feature space granularity: this can hold well in low-dimensional spaces, but as we move to extremely high-dimensional spaces like those generated by images [8] one can simply see that this strategy cannot be pursued. What happens instead is that a training dataset manages to represent the underlying generative distribution with a certain granularity and different identically sized datasets lead to significative diverse reconstructions of the underlying probability distribution. We should then shift the objective of the training task from *indefinitely optimizing the performance metric on the training dataset* to *optimizing enough to reproduce the general trends of the data, but not enough to reproduce the granularity effects of the particular sampling of data-generating distribution.* The situation of failing to reproduce general traits of the data-generating distribution is called **underfitting** while **overfitting** describes the case in which the model starts to optimize around sample-related specific traits and cannot generalize to other distribution samplings.

A basic but essential strategy to avoid overfitting is using two different datasets in the training phase: a *training set* which we use as the main proxy for the data-generating distribution and a *validation set* against which we validate model performance. Assuming that these two datasets are indipendently extracted and identically distributed, the optimization goal becomes lowering the train and validation errors while ensuring that the difference between the two stays low.

Measuring performance metrics on both datasets during an optimization run will show that performance improves for both datasets until the model starts to reproduce the peculiarities of the training dataset: in this case we observe that the train performance continues to slowly decrease while the validation error increases as the model starts to overfit: interrupting the training process before the validation error starts to increase should ensure that the model optimizes while retaining enough generalization capabilities, this trick is called *early stopping*.

[8] Monochrome 8-bit 512x512 images have 262.144, pixels each with 255 possible gray levels, even if we assume that all non-random images live in a lower-dimensional variety of this space, uniform sampling is simply not feasible.

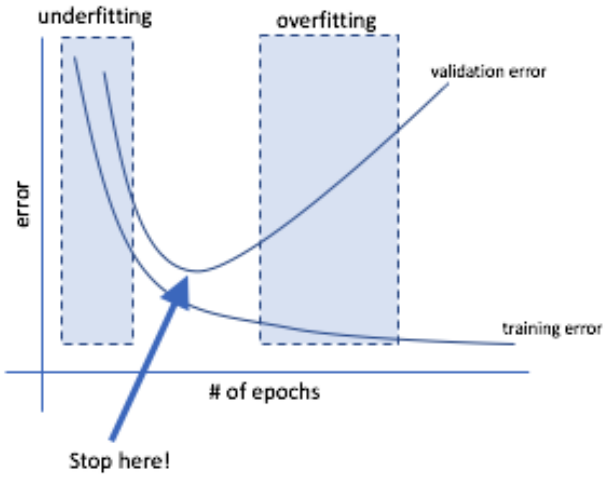

Figure 1.4: Overfitting and Underfitting

Of course, when evaluating the absolute performance of the model, neither the training set nor the validation set can be used because of the implicit statistical correlation: a third *test dataset* shall be used to evaluate performances in a reproducible way.

Another key factor in generalization capabilities is given by *model capacity*. The particular choice of the model and the number of its degrees of freedom determines its *representational capacity*. Large model complexities can expose the model to overfitting, the concept is better understood by looking at figure 1.5: fitting three different polynomial models to the data we easily see that using simplistic models the distinctive features of the data are not captured, and introducing too many parameters reduces the training error at the cost of loss of generalization.

Using, again, a training set and a validation set we can reveal a behavior that closely resembles figure 1.4: training models with increasing capacities lowers the training errors but too high model complexities cause the models to inevitably overfit.

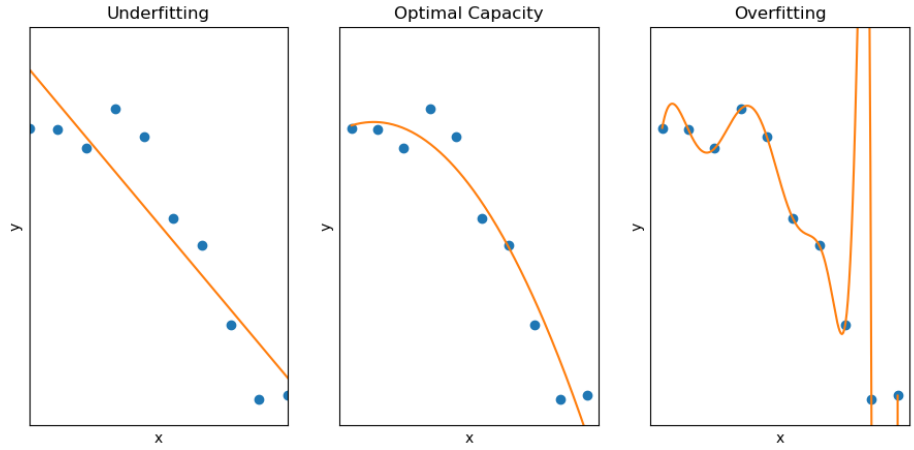

Figure 1.5: Model Complexity

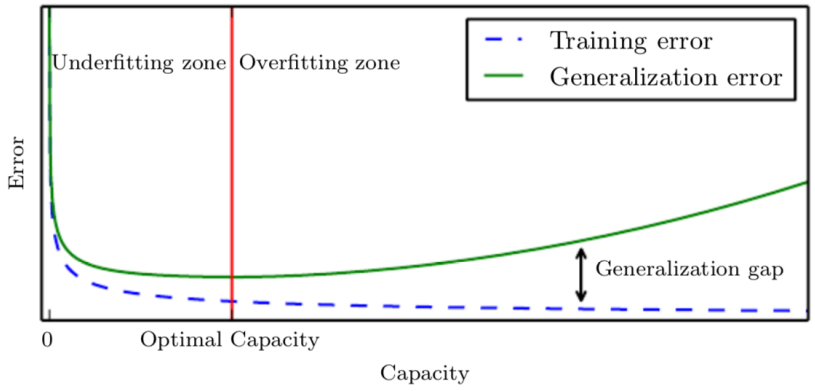

Figure 1.6: Model Capacity and Training/Validation Errors [41]

1.5 Convolutional Neural Networks

Among all the possible Machine Learning models, this work focuses on a particular subset called *Convolutional Neural Networks (CNN)*. CNNs are a particular type of the *Artificial Neural Networks* we introduced in 1.1.1 and can basically be considered graphs of operations with a layered structure and tunable free parameters, they have in input layer, one or more hidden layers and one output layer that, in classification tasks, has the same cardinality as the number of classification classes, or, in the

segmentation case, can form an image. The main new element that CNNs introduce is the use of the *convolution* operation with free parameters as layers in the network's computation graph: while layers in MLPs are *fully connected* - i.e. every neuron of a given layer shows as a weighted input for every neuron in the next layer (we shall see a more fitting definition in section 1.5.4) - in CNNs feature maps are obtained using sequential *convolution* operations with trainable kernels.

1.5.1 Convolutional Layers

The *convolution* concept is ubiquitous in Physics, Mathematics, Engineering and any scientific or technical field so there are high chances that it is already be familiar to the reader but for clarity we shall provide a quick and concise definition anyway. We define the convolution operation between two functions $x(t)$ and $w(t)$ as

$$s(t) = (x * w)(t) = \int_{-\infty}^{+\infty} x(a)w(t-a)da \tag{1.26}$$

This definition is valid if both $x(t)$ and $w(t)$ are real-valued functions defined on a continuous index, but when dealing with digital images all we have at hand are discrete-valued functions. Convolution operation is simply defined for these discrete functions as well :

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{+\infty} x(a)w(t-a) \tag{1.27}$$

When passing from one to two dimensions, with $X(i,j)$ and $W(i,j)$ being our two-dimensional functions, we can easily generalize definition 1.27 as

$$S(i,j) = (X * W)(i,j) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X(m,n)W(i-m, j-n) \tag{1.28}$$

Convolutions benefit from commutative property: this means that the expression 1.28 is totally equivalent with the case in which the operator order is inverted

$$S(i,j) = (W * X)(i,j) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X(i-m, j-n)W(m,n) \tag{1.29}$$

Looking at the indexes inside the summations we see that this is possible because the two images are read in opposite order: one image is "flipped" with respect to the

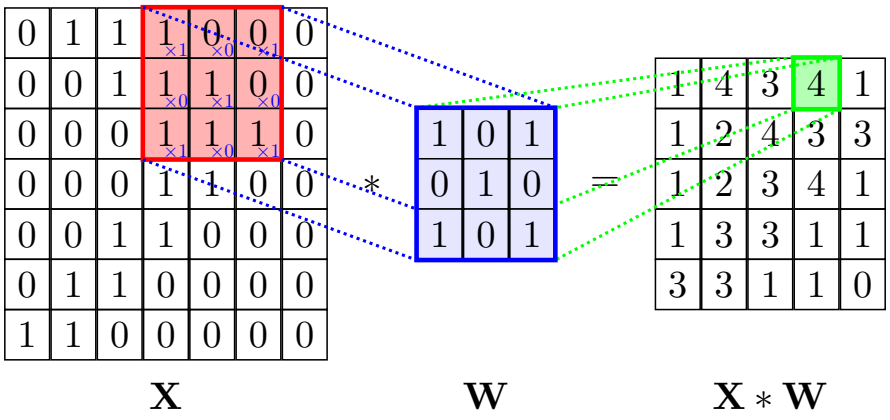

Figure 1.7: Convolution Operation

other. While this kind of property is mathematically interesting, on the implementation side it doesn't give any significant advantage and flipping one of the two images adds to the computational cost without any practical return on the learning process. Computationally speaking it's more convenient to implement *cross-correlation* instead: *cross-correlation* is conceptually very similar to *convolution* but avoids flipping one of the two images:

$$S_{\text{cross-correlation}}(i,j) = (W * X)(i,j) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X(i+m, j+n) W(m,n) \quad (1.30)$$

Convolutional Neural Networks should then be called *Cross-Correlation Neural Networks*, nevertheless they mantain the *convolutional* term for association purposes with the very familiar convolution concept as opposed to the lesser known cross-correlation.
Discrete convolutions can actually be viewed as matrix multiplications with a particularly constructed matrix, more on this topic will be mentioned in appendix B but for now it's enough to say that the possibility of calculating convolutions using basic multiplications represents a major computational advantage as it means that they can be evaluated in an heavily vectorized and efficient form.

Neural Networks can be built using the convolution concept if, starting from one input layer, we define the next layer as the result of a convolution with a weight image whose matrix values, $W(i,j)$ are free parameters of which we learn the values

using Gradient Descent or any loss minimizer. In order to preserve non-linearity and representational capabilities of the model (a small note on that in sec. 1.39), outputs of the convolution operation can be passed through a nonlinear *activation function* like $ReLU$ (Rectified Linear Unit) defined as

$$ReLu(x) = \max(0, x) \tag{1.31}$$

which breaks linearity and inhibits negative responses of the network. Other non-linear monotonic functions can be used for activation purposes such as *sigmoid* functions or *hyperbolic tangents*.

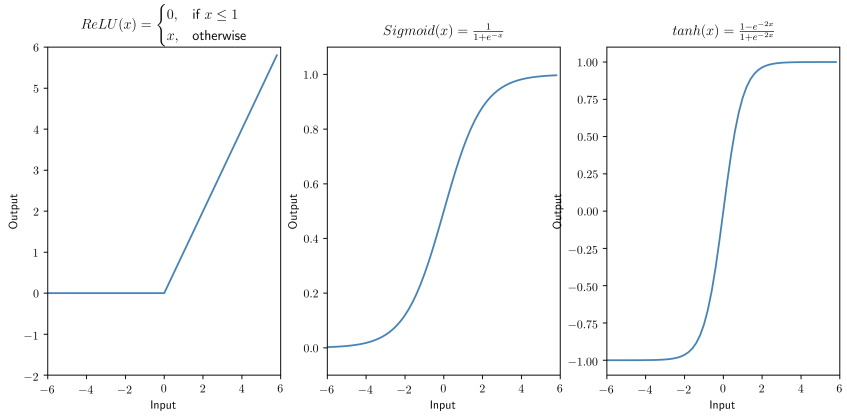

Figure 1.8: Activation functions

Networks built in this way feature very interesting properties. Firstly, instead of learning a set of weights that scales with image dimension, we're learning a small fraction of this set by sharing most of these parameters, in the process we gain a *shift invariance* property that is particularly useful in classification tasks. Another interesting property is the fact that convolution filter response is intrinsecally local (response of a neuron is directly influenced only by a small number of near pixels in the input image, called *receptive field*), this makes CNNs particularly good at identifying patters in images.

1.5.2 Padding

The 2D discrete convolution operation, as described in 1.5.1, takes an $n \times n$ input X, a $f \times f$ filter W and outputs an $n - f + 1 \times n - f + 1$ image: the reason of this is

easily understood by looking at figure 1.7 and noticing that there are only $n-f+1$ possible positions of the kernel on the image that can produce an output.
This property highlights two main issues: in the first place, if we have several convolutional layers, the output size contraction is going to noticeably reduce the inputs's sizes after a few layers, which is an effect we may want to avoid if the network is particularly deep, secondly, if the kernel slides sequentially across the inputs, border pixels are going to figure in the convolution computation way less than the central ones as the sliding window is going to cover them in fewer occasions.

To overcome both problems the inputs can be extended by adding p zeros to their borders before convolution so that the new outputs are $n-f+1+2p \times n-f+1+2p$ sized. There are three types of padding that are generally used in convolutional nets, known as *full padding*, *same padding* and *valid padding*. *Valid padding* just sets $p=0$ so that the outputs are $n-f+1$ and represents the same exact operation as convolution without padding, In *full padding* we want every possible complete or partial superposition of the kernel to the inputs to be taken into account in the outputs so that outer pixels of the inputs no longer result as less important in the final output: this can be done by extending the input image in such a way that the new size is $n+f-1$, implying $p=f-1$. In the *same padding* case we want the new outputs to have the same dimensionality as the inputs, meaning that

$$\begin{aligned} n+2p+f+1 &= n \\ p &= \frac{f-1}{2} \end{aligned} \tag{1.32}$$

for same padding to be employed we need the filters size to be odd.

To better visualize the concept let's consider the convolution of a 5×5 sized input X with a 3×3 filter W:

$$X = \begin{pmatrix} 27 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix} \qquad W = \begin{pmatrix} 1 & 3 & 1 \\ 0 & 5 & 0 \\ 2 & 1 & 2 \end{pmatrix} \tag{1.33}$$

The different paddings of X are

$$X_{\text{padded}} = \begin{pmatrix} \text{full} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & \text{same} & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & \text{valid} & & & & & & \\ 0 & 0 & 27 & 24 & 1 & 8 & 15 & 0 & 0 \\ 0 & 0 & 23 & 5 & 7 & 14 & 16 & 0 & 0 \\ 0 & 0 & 4 & 6 & 13 & 20 & 22 & 0 & 0 \\ 0 & 0 & 10 & 12 & 19 & 21 & 3 & 0 & 0 \\ 0 & 0 & 11 & 18 & 25 & 2 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{1.34}$$

The resulting convolutions then are

$$X * W = \begin{pmatrix} \text{full} \\ 17 & 75 & 90 & 35 & 40 & 53 & 15 \\ & \text{same} & & & & & \\ 23 & 159 & 165 & 45 & 105 & 137 & 16 \\ & & \text{valid} & & & & \\ 28 & 198 & 120 & 165 & 205 & 197 & 52 \\ 56 & 95 & 160 & 200 & 245 & 184 & 35 \\ 19 & 117 & 190 & 255 & 235 & 106 & 53 \\ 20 & 89 & 160 & 210 & 75 & 90 & 6 \\ 22 & 47 & 90 & 65 & 70 & 13 & 18 \end{pmatrix} \tag{1.35}$$

1.5.3 Pooling Layers

Another fundamental building block of CNNs is the *Pooling* layer: pooling is a form of down-sampling of the original image. What, an intuitive level, a pooling layer does, is to replace the original image with a summarized version of it by dividing it in subsections and applying a selection rule, being it linear or nonlinear. The most used pooling layer is the the *max-pooling* that outputs the maximum of a selected cell.

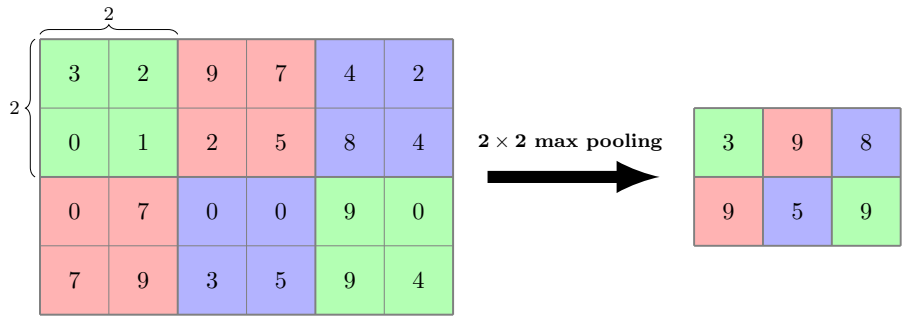

Figure 1.9: Pooling Operation

In figure 1.9 we can see an example of *max-pooling* in which the original image is divided in 2x2 cells and from each cell only the highest value is selected, creating a downsampled image with a rough map of max activations.

Other pooling layers include *average pooling* and *mean pooling*. Pooling operations are useful in reducing the number of parameters and model complexity and, therefore, controlling overfitting, especially in classification tasks. Most pooling layers are approximatively invariant to small transformations of the inputs and help the network generalize classification task with almost-scale-invariant internal representations.

1.5.4 Fully Connected Layers

The connectivity scheme of MultiLayer Perceptrons is not completely discarded in Convolutional Neural Networks and, actually, creating connections between every neuron of two successive layers is often necessary in classification tasks where the internal representations of the data have to be linked to a small number of output classes, in these scenarios *Fully Connected* or *Dense* layers make excellent classification steps.

A fully-connected layers is generally implemented as an affine transformation followed by a nonlinear element-wise function. The affine transform

$$\vec{z}(\vec{x}) = W^{\intercal}\vec{x} + \vec{b} \tag{1.36}$$

is a simple transformation of the same kind we would have found in a Perceptron. It's important to consider that the composition of two or more affine transformations

still is an affine transformation: let's try to compose two of these transforms

$$\begin{aligned} \vec{z}(\vec{x}) &= W^\intercal \vec{x} + \vec{b} \\ \vec{s}(\vec{y}) &= M^\intercal \vec{y} + \vec{d} \end{aligned} \tag{1.37}$$

without introducing a nonlinearity in the composition the result will always be an affine transformation

$$\begin{aligned} \vec{t}(\vec{x}) &= \vec{s}(\vec{z}(x)) \\ &= M^\intercal \vec{z} + \vec{d} \\ &= M^\intercal (W^\intercal \vec{x} + \vec{b}) + \vec{d} \\ &= M^\intercal W^\intercal \vec{x} + M^\intercal \vec{b} + \vec{d} \\ &\equiv T^\intercal \vec{x} + \vec{k} \end{aligned} \tag{1.38}$$

from a representational point of view stacking more of these layers would bring no advantage as no change to the space of functions that this operation could define is made. When nonlinearity is introduced in the form of an element-wise transformation $g(\vec{x})$ the resulting transform is no longer affine and con be combined in a representationally meaningful way.

$$\vec{z}(\vec{x}) = g(W^\intercal \vec{x} + \vec{b}) \tag{1.39}$$

Since $g(\vec{x})$ is an element-wise transform, $z(\vec{x})$ can still be expressed in an easy matrix form

$$\begin{aligned} \vec{z}(\vec{x}) &= g(W^\intercal \vec{x} + \vec{b}) \\ &= g\left(\begin{bmatrix} W_{11} & W_{12} & \dots & W_{1K} \\ \vdots & \vdots & & \vdots \\ W_{K1} & W_{K2} & \dots & W_{KK} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_K \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix} \right) \\ &= g\left(\begin{bmatrix} W_{11}x_1 + b_1 & W_{12}x_2 + b_2 & \dots & W_{1K}x_K + b_K \\ \vdots & \vdots & & \vdots \\ W_{K1}x_1 + b_1 & W_{K2}x_2 + b_2 & \dots & W_{KK}x_k + b_k \end{bmatrix} \right) \\ &= \begin{bmatrix} g(W_{11}x_1 + b_1) & g(W_{12}x_2 + b_2) & \dots & g(W_{1K}x_K + b_K) \\ \vdots & \vdots & & \vdots \\ g(W_{K1}x_1 + b_1) & g(W_{K2}x_2 + b_2) & \dots & g(W_{KK}x_k + b_k) \end{bmatrix} \end{aligned} \tag{1.40}$$

Dense layers are way *heavier* than Convolution layers in the sense that they introduce extremely high number of free parameters: it's not unusual that the vast majority of a network's degrees of freedom are located in its dense layers. Not every task needs this kind of connectivity and networks that don't have Dense layers are called *Fully Convolutional Neural Networks*[9] and there's an entire branch of segmentation tasks that employ these networks.

1.6 Regularization Strategies

A major problem in training Convolutional Neural Networks, as mentioned in 1.4, is overfitting: in a training/test set setup this translates as good performance on the test set and large errors on the test set. The main goal of the training process is producing models that perform well on previously unseen data, or, in other terms, reducing overfitting.
Regularization is a collective name that refers to all the strategies that aim to reduce test errors, usually at the cost of higher errors in the training dataset. There are many different approaches in reaching this goal that look at the problem from different perspectives: in this section we introduce some of the most common ones.

1.6.1 L1 and L2 Regularization

As seen in 1.4, a primary cause in overfitting is model complexity: too many free parameters make the model prone to overfitting. A possible way to look at the problem would be finding a way to let the training process preferrably choose simpler models. This can be done by introducing terms in the loss function that penalize complexity in the model making simple structures artificially preferrable. Model complexity is a function of the number of nonzero weights: if a weight is zero the link's contribution is null and the specific term controlled by that weight doesn't effectively add to its effective capacity. Inserting a term in the loss function that specifically penalizes high weight values should make "unuseful" link weights vanish to zero, this is exactly what happens in L1 and L2 regularization strategies.

[9] In section 2.1.2 we will see that the classification of architectures either as fully-convolutional or non-full-convolutional is indeed a false dichotomy as dense layers can be interpreted as a particular case of convolutional layers.

The new loss functions will then be in the form

$$\tilde{J}(w, X, y) = J(w, X, y) + \alpha\Omega(w) \tag{1.41}$$

where $J(w, X, y)$ represents the original loss function, w represents model weights and the term α is a regularization hyperparameter.

In L2 regularization the additional term is the ℓ_2 norm of the weight vector

$$\Omega(w) = \frac{1}{2}\|w\|_2^2 \tag{1.42}$$

so that the actual loss function becomes

$$\tilde{J_{L_2}}(w, X, y) = J(w, X, y) + \alpha\frac{1}{2}w^\intercal w \tag{1.43}$$

The loss function remains differentiable and its gradient is

$$\nabla_w \tilde{J_{L_2}}(w, X, y) = \nabla_w J(w, X, y) + \alpha w \tag{1.44}$$

1.44 can be interpreted as an iterative subtraction of a certain portion of w proportional to α, this encourages small weight values but doesn't force weights to be exactly zero, thus mantaining model complexity, if not in an effective sense, at least in the storage needed to represent the model. It would be ideal to force to exactly zero those weights that are too small to influence the model and enforce model sparsity in those cases: this can be done using L1 regularization.

In this case the regularization term is the ℓ_1 norm of the weight vector

$$\Omega(w) = \|w\|_1 = \sum_i |w_i| \tag{1.45}$$

the loss function becomes

$$\tilde{J_{L_1}}(w, X, y) = J(w, X, y) + \alpha\|w\|_1 \tag{1.46}$$

which has a corresponding gradient

$$\nabla_w \tilde{J_{L_1}}(w, X, y) = \nabla_w J(w, X, y) + \alpha\,\mathrm{sign}(w) \tag{1.47}$$

In this case we have that the iterative weight subtraction has a fixed step, but the whole loss function doesn't remain differentiable in $w = 0$, this makes L1 regularization incompatible with simple gradient descent, even if modern optimizers can handle piecewise continuous functions.

1.6.2 Data Augmentation

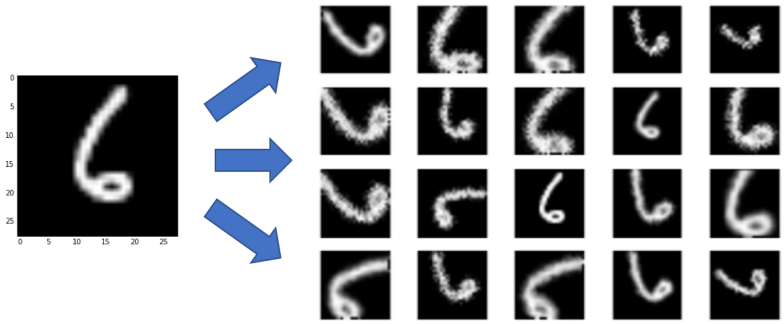

Figure 1.10: Data Augmentation

Another perspective on the problem is that of the training dataset's limitedness: small datasets don't cover much of the data-generating distribution's space of definition. Fortunately, in most cases, this problem can be tackled by simulating feasible transformations of the input data to generate new likely inputs for the model. Of course the kind of allowed transformations depends on the expected symmetries of the dataset: when segmenting cars on a dataset for driving automation it's perfectly reasonable to generate new artificial data points by horizontally mirroring original images[10] but not doing the same operation on the vertical axis as it's very unlikely that the original distribution contains many examples of upside-down cars, same goes for 90 degree rotations as we don't expect in our daily commute to the Phyisics Department to find many examples of Teslas in perfect vertical balance on their front or on their back. Another thing one has to avoid is applying transformation that - in classification cases - would lead to class change: when designing a character recognition system, rotations of the "d" and "b" characters would effectively change

[10]One could argue that the car's commands asymmetry would produce non-valid augmented inputs: considering that an horizontal flip of a left-driving car would result in a right-driving car (and vice-versa) and that a driving automation company is not likely to produce two different machine vision systems for left-driving and right-driving countries, I'd happily dismiss this argument as non practically relevant.

their class. Possible transformations space from rotations and reflections to translations[11], color shifts, noise introduction, elastic transformation: every transformation that produces a likely new data input is to be used.

1.6.3 Dropout

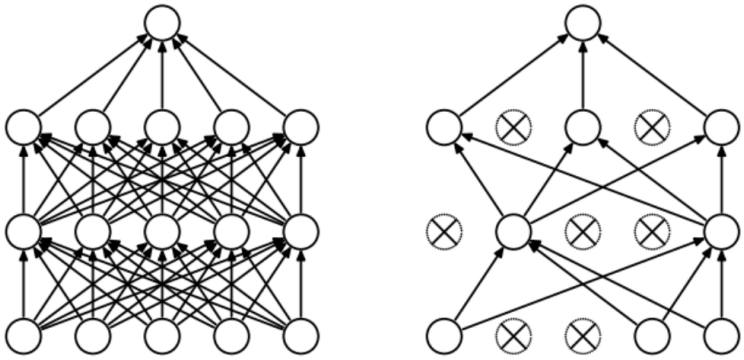

Figure 1.11: Dropout [112]

An interesting and extremely effective technique in regularization is *dropout*: the main idea behind dropout is that of the random exclusion of a number of weights from a training step, at each training step, every node is either "dropped" with a probability p or "kept" with a probability $1 - p$, in the case of drop both the node and its connecting edges are temporarily removed from the model and the training happens on the remaining neurons. The intuitive reason of why this process works is because it breaks the network's tendency to overly specialize certain weights: during the training, some of the weights tend to be assume critical roles with neighboring neurons becoming more and more reliant on their activations, this tendency in over-specialization can make the model fragile when exposed to new data. By randomly deactivating some of the neurons, the neighboring nodes have to step in and activate to maintain correct predictions. This makes the network less sensitive to locally specific weights and is believed to create multiple concurrent internal representations that help the model generalize to unseen data.

[11] Even models that are designed to be spatially invariant benefit from small translations in data augmentation phase [41]

1.7 Other Methods in CNN Training

1.7.1 BatchNormalization

A typical issue in training deep neural networks is known as the *Internal Covariate Shift* problem. In deep neural models with multiple stacked layers we can view the outputs of each layer as composite functions of the outputs of the previous layers, in an iterative way, up to the inputs, so that the outputs of a certain layer depend on the parameters of all the ones up in the computation graph. The obvious consequence is that minor changes in the parameters of a layer tend to propagate and amplify in the following ones, more and more prominently as the network gets deeper.
To better focalize the problem let's imagine we have a network made of only two layers F_1 and F_2, each with a single parameter, respectively θ_1 and θ_2. The network parameters are learned to minimize a loss

$$\ell = F_2(F_1(u, \theta_1), \theta_2) \tag{1.48}$$

where u is the network's input. The F_2 layer is ideally equivalent to a stand-alone network that computes ℓ using $x = F_1(u, \theta_1)$ as inputs so that

$$\ell = F_2(x, \theta_2) \tag{1.49}$$

Applying a simple gradient descent like the one in 1.3.1 to the two-layer network we can update θ_2

$$\theta_2 \leftarrow \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(x_i, \theta_2)}{\partial \theta_2} \tag{1.50}$$

where m is the batch size and α the learning step. Notice that the 1.50 update step is perfectly equivalent to what we would have written considering the F_2 network alone, with x inputs.

It's rather intuitive to see how the learning process for the F_2 standalone network would be easier if the distribution of x wouldn't change at every update step of θ_1. The problem of the changes in the distribution of network activations due to the change of network parameters during training is known as the *Internal Covariance Shift* problem and it has been known for a long time.

In 2015 Szegedy et al. introduced an innovative method to tackle the ICS problem: the main idea behind it is that, for the model to correctly train its weights, the

distribution of input network activations cannot -obviously- be fixed but for a given layer one could transform the outputs of preceeding ones in such a way that both their mean and variances are fixed. In order to do such transformation, ideally, one should know in advance all the activation distributions related to the entire dataset: this is extremely impractical, thus the mean and variance are estimated using the mini-batch of the training process as a proxy for the full distribution.

Let us denote B as a training mini-batch of activations, with size m. The mean and variance of B are defined as

$$\begin{aligned} \mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_b)^2 \end{aligned} \tag{1.51}$$

We can normalize separately each dimension of the activations for a layer with a d-dimensional input $x = (x^{(1)}, \ldots, x^{(d)})$, with respect to μ_B and σ_B

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{{\sigma_B^{(k)}}^2 + \epsilon}} \tag{1.52}$$

with $k \in [1, d]$, $i \in [1, m]$ and $\mu_B^{(k)}$ ${\sigma_B^{(k)}}^2$ are the k-components of respectively respectively the mean and variance.

The resulting distribution (ignoring ϵ which is just a small finite term for numerical stability) is a characterized by zero mean and unitary variance.

Of course constraining a layer's outputs to have null mean and unitary variance could limit its representational capacity: to overcome this, two additional learnable parameters are added in a second transformation step

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)} \tag{1.53}$$

by learning $\gamma^{(k)}$ and $\beta^{(k)}$ the model restores some of the representational capacity that has been lost after normalization. It's important to notice that, while the second transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}(\hat{x}_i^{(k)})$ is passed to the subsequent layers, the normalized output $\hat{x}_i^{(k)}$ remains internal to the current layer.

Chapter 2

CNNs for Segmentation

In Convolutional Nets, there is no such thing as "fully-connected layers".

Yann LeCun
comment on his Facebook page

2.1 State of the art in 2D Semantic Segmentation

This work aims to use a particular kind of Convolutional Neural Network to solve a *semantic segmentation* task, as defined in 1.2.2: we want to train a model that, given an image, returns us a per-pixel classification of it. There are many different approaches to this kind of task, each with its own perks and advantages and each facing the problem from slightly different angles, the specific approach used in this work is based on the so-called U-NETs, which we'll introduce a few paragraphs below.

2.1.1 The Sliding Window Approach

Down at its core semantic segmentation is just a classification task repeated for every single pixel of an image: a possible solution path for acccomplishing this kind of task would be tracing back the segmentation problem to a single-pixel classification tasks, repeated for the total number of pixels in the image. This can be done by using a "sliding window" approach: in its classical setting a CNN classifier is trained to

express the class membership probability of the central pixel in a small patch: the input image can be divided into overlapping patches (with opportune padding at the borders) and fed into an array of classifiers to obtain a matrix of predictions which we can use make out a segmented version of the original image. A naive version of this kind of approach with an instance of the same model being deployed for each classification patch comes with great computational expense, although many optimizations are possible so that the computational cost of the algorithm does not scale linearly with the number of pixels.

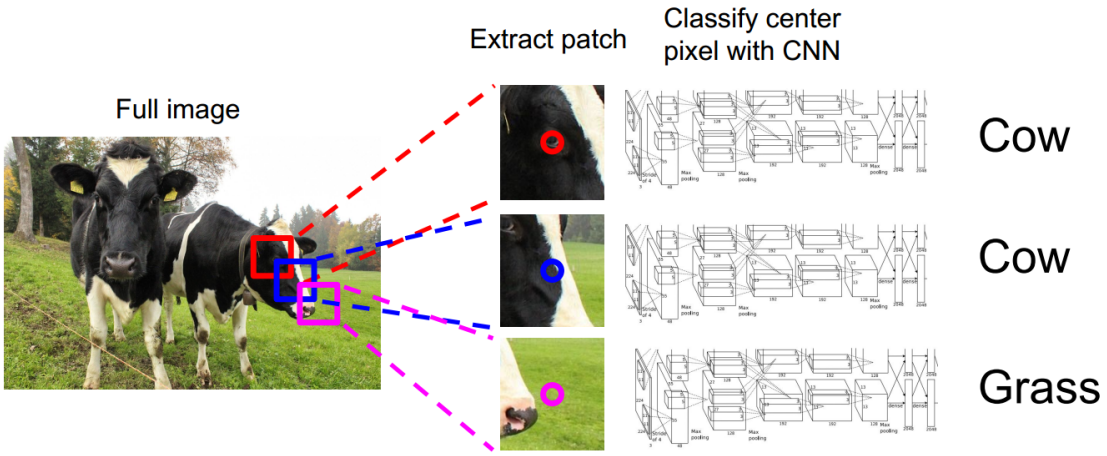

Figure 2.1: Semantic segmentation using sliding windows [55]

2.1.2 Convolutional Interpretation of Sliding Windows

The most concerning aspect of employing an entire instance of a neural network to classify an image, one window at the time, is the computation redundancy in the overlapping regions. The process of sequentially sliding a window across an input plane should trigger an intuitive association to section 1.5.1 where we introduced the concept of convolutions. Convolutional layers are fundamentally just very efficient implementations of the convolution with a sliding kernel operation we just introduced (in appendix B we will actually see that discrete convolution can actually be expressed as a much simpler and more intuitive matrix multiplication), making it very similar to the sliding operation used for segmentation. As observed in Sermanet, LeCun et. al - "Overfeat: Integrated Recognition, Localization and Detection using

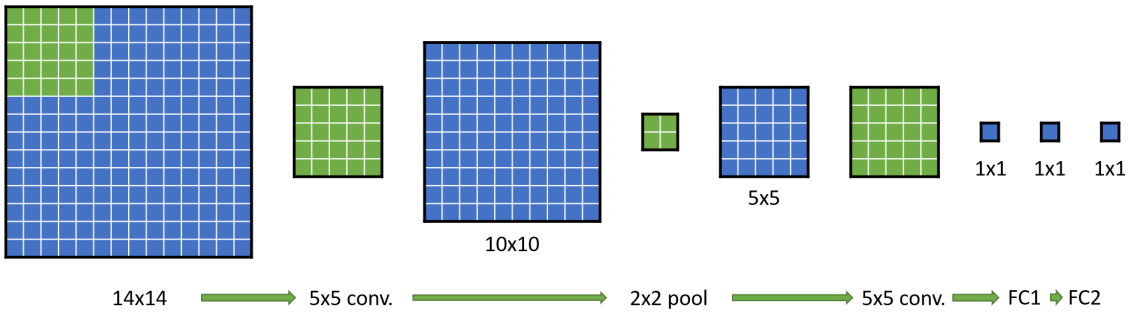

Figure 2.2: CNN Classifier with FC Layers, adapted from [104]

Convolutional Networks" [104], this similarity can be used to evaluate responses over an extended images by exploiting the inherent efficiency of convolutions.

Let's suppose we have a CNN classifier like the one in figure 2.2. Temporarily omitting depth dimensions for both feature maps and filters and just mentioning the "spatial" dimensions, the figure describes a network that accepts 14×14 sized inputs, has a feature extracting stage made of a 5×5 convolution followed by 2×2 maxpooling that creates a 5×5 feature map, which we convolve with a 5×5 kernel to obtain a single-pixel-wide response (depth dimensions are omitted in this representation but it shall be clear that response is actually an array of pixels which dimension depends on the actual number of employed 5×5 convolution filters), this single-pixel layer, lastly, undergoes a final classification stage with two fully-connected layers to produce the final output response.

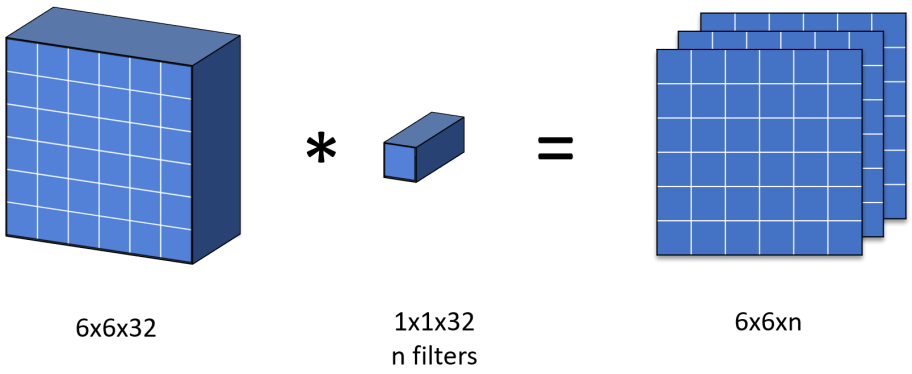

Figure 2.3: 1x1 Convolution

In "Network in Network" by Lin et al. [73] it's suggested that fully connected layers can actually be viewed as $1 \times 1 \times d$ convolutions : when convolving with $1 \times 1 \times d$ filters each $1 \times 1 \times d$ slice of the input matrix undergoes a dot product with the $1 \times 1 \times d$ filter, so that a $1 \times 1 \times d$ convolution of a $n \times n \times d$ input results in a $n \times n$ output and, by changing the number of filters, different output shapes can be obtained.

Making a numerical example, let's see how we can reformulate a fully connected layer as a convolution step. We start from the fully connected layer $f(x)$ defined by a weight matrix W and a bias vector $\vec{b}$:

$$\begin{aligned} f(x) &= ReLU(W \cdot \vec{x} + \vec{b}) \qquad \vec{x} \in \mathcal{R}^3 \\ W &= \begin{bmatrix} 0 & 1 & 1 \\ 2 & 3 & 5 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} 8 \\ 13 \end{bmatrix} \end{aligned} \tag{2.1}$$

The connection matrix W with its bias vector $\vec{b}$ can actually be seen as two different $1 \times 1 \times 3$ convolution kernels, each with a single bias value

$$\begin{aligned} \vec{w_1} &= \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \qquad b_1 = \begin{bmatrix} 8 \end{bmatrix} \\ \vec{w_2} &= \begin{bmatrix} 2 & 3 & 5 \end{bmatrix} \qquad b_2 = \begin{bmatrix} 13 \end{bmatrix} \end{aligned} \tag{2.2}$$

Each convolution can then be computed using the 1.30 definition -notice that we use the cross-correlation definition instead of the *proper* convolution one- and the bias value is added afterwards, so that

$$f_{\text{conv}}(\vec{x}) = ReLU\left(\begin{bmatrix} [0,1,1] * \vec{x} \\ [2,3,5] * \vec{x} \end{bmatrix} + \begin{bmatrix} 8 \\ 13 \end{bmatrix} \right) \tag{2.3}$$

If we use as a numeric input value

$$x = [1,2,3] \qquad \in \mathcal{R}^3 \tag{2.4}$$

and index the elements of $\vec{x}, \vec{w_1}, \vec{w_2}$ as $-1, 0, 1$, we can apply definition 1.30

$$\begin{aligned} \vec{w_1} * \vec{x} &= \sum_{j=[-1,0,1]} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} [j] \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} [j+0] = 5 \\ \vec{w_2} * \vec{x} &= \sum_{j=[-1,0,1]} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} [j] \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} [j+0] = 23 \end{aligned} \tag{2.5}$$

so that the convolution layer computes

$$f_{\text{conv}}(\vec{x}) = ReLU\left(\begin{bmatrix}5\\23\end{bmatrix} + \begin{bmatrix}8\\13\end{bmatrix}\right) = ReLU\left(\begin{bmatrix}13\\36\end{bmatrix}\right) \tag{2.6}$$

Which is exactly the same result we would have obtained via dot product:

$$f(\vec{x}) = ReLU\left(\begin{bmatrix}0 & 1 & 1\\2 & 3 & 5\end{bmatrix} \cdot \begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}8\\13\end{bmatrix}\right) = ReLU\left(\begin{bmatrix}13\\36\end{bmatrix}\right) \tag{2.7}$$

Now that we established that fully connected layers can be interpreted as 1×1 convolutions we can apply this knowledge to reformulate the sliding windows problem. Looking at the upper part of figure 2.4 we see the same network as 2.2 but now the fully connected layers have been replaced by 1×1 convolutions. If we want to

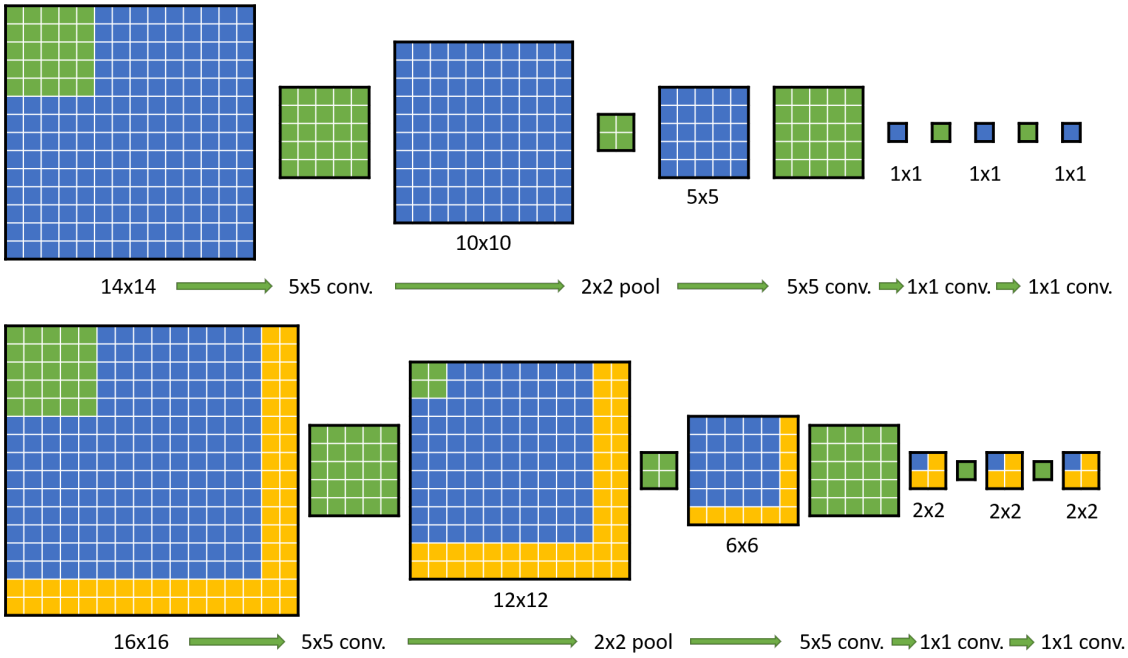

Figure 2.4: Efficient Computation of Sliding Windows, adapted from [104]

calculate the response of the same network on a wider, image with 16×16 pixels instead of 14×14, using the sliding window approach we would create a 14×14 window and slide it over the 16×16 image to get a 3×3 output image - if using a

1 pixel stride - or a 2×2 image when using a 2 pixel stride.
A way to drastically increase efficiency by sharing most of the calculation between overlapping regions is to use the same network, with the same convolutions and weights, directly on the 16×16 image [104]. The first convolution layer produces a 12×12 feature map, which dimensions are halved to 6×6 by the pooling layer, the 5×5 convolution then does not produce a 1×1 input but a 2×2 one and the 1×1 convolutions mantain the same space dimensions to the network output. The same identical network does not produce a single pixel anymore but a 2×2 response map where each pixel corresponds to the network's response at each of the four positions when sliding a 14×14 window with stride 2. If the network is made only of convolution and pooling steps the sliding window stride is determined by the number and type of pooling stages, this introduces the concept of a *native stride* of the classifier as the stride at which the convolutional network is going to produce output maps when used on extende images.

A classifier network with its fully connected layers reformulated as 1×1 convolutions can be thought to act as a single filter with an associated s_{native} stride that is 1 when no poolings are involved and whose outputs have size

$$O = \frac{W - F}{s_{\text{native}}} \tag{2.8}$$

where W and F are, respectively, the image and filter sizes.

A way to produce outputs that have the same size as the inputs is counter the native stride by artificially upsampling the image before feeding it to the network using, for example, a bilinear or nearest neighbor interpolation. Depending on the specific application and data characterization this can possibly produce noticeable artifacts but makes it possible to have 1:1 responses using just a classifying network.

2.1.3 Fully Convolutional Networks

In order to avoid image shrinking one could imagine a network that consists only of padded convolution layers that preserve the image size across its whole extent, this kind of network would not need multiple passes for a single prediction and wouldn't have fully-connected layers. On the other side a model made only large-scale convolutions would still be computationally expensive and, unlike traditional

classifying CNNs, its architecture wouldn't structurally push the training towards the abstraction of increasingly complex features in the last low dimensional layers. CNNs usually extract complex features from the inputs by reducing their dimensionality across the network, this *downscaling* is done using both strided convolutions and pooling layers, resulting in summarized versions of the features that are mapped to a classification class using one or more fully connected layer. The general idea behind the use of low dimensional internal representations to make the network abstract complex features from large inputs can be used to build fully convolutional segmentation networks by introducing upscaling trainable layers that increase dimensionalities instead of decreasing them. These networks would be formed by two different stages: an "encoding" one that extract a low dimensional representation of the inputs and a "decoding" stage that uses upscaling layers to recreate high-resolution output segmentations from low-dimensional internal representations of the inputs.

2.1.4 Unpooling and Transposed Convolution

To make the "upscaling" part of the network we need layers that were not introduced in Chapter 1. There are many image processing methods to obtain high-resolution images from their low-resolution versions that involve some kind of interpolation policy to assign a value to the extra pixels, such as *nearest neighbor* rules or *bilinear interpolation*. These methods do not have trainable free parameters and always produce the same results, meaning that their behaviour is not going to be optimized during training . The simplest upsampling method is *Unpooling*: unpooling can be viewed as an approximate inverse of the "pooling" operation and creates an high resolution version of the input image by placing values in fixed locations, the remaining pixels can be set to zero -producing what's called a *bed of nails* unpooling- or can be filled in using a nearest neighbor policy. The new positions for the pixels are not necessarily hard-coded: if the network is symmetrical, the pixels positions from a pooling layer in the downscaling segment can be used in the corresponding unpooling layer in the upscaling part to retrieve some of the compressed spatial information.

Although it's very common - and often produces good results - to couple a "classic" deterministic upsampling method with a convolutional layer to mimic "trainable"

Figure 2.5: Unpooling

upsampling, there is a way of formulating such a layer as a convolutional layer. *Transposed Convolution* can be used as a trainable filter for upsampling. The general idea to comprehend the function of a transposed convolution layer is that it enables us to go in the "opposite direction" of a normal convolution: from a tensor with the shape of the convolution output to another that has the shape of the inputs. In fully connected layers this can easily be achieved by using a weight matrix with transposed shape, but it's not trivial to get the same functionality with convolutions. The key to the problem resolution is that discrete 2D convolutions can actually be expressed as matrix multiplications using Toeplitz matrices, as thorougly explained in appendix B. Making a simple example in one dimension, we consider a convolution between two vectors $\vec{x} = [x, y, z]$ and $\vec{a} = [a, b, c, d]$ and convert one of the inputs to a Toeplitz matrix, such that their convolution can be rewritten as

$$\vec{x} * \vec{a} = X\vec{a} = \begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ax + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix} \tag{2.9}$$

The *transpose convolution* operation can be created from 2.9 simply by transposing

the first term and using a slightly different padding scheme for the second one

$$\vec{x} *^{\intercal} \vec{a} = X^{\intercal}\vec{a} = \begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix} \tag{2.10}$$

This formulation is particularly useful because the so-obtained convolution transposes are actually just regular convolutions with the only difference of the padding rules. The procedure for 2D and 3D matrices is not conceptually different and requires the use of block Toeplitz matrices.
Layers built this way are generally known in literature as *UpConvolution* or *Deconvolution* layers: the *Deconvolution* name is misleading because it may suggest *inverse convolution*, which is not what these layers do, the general inspiration for this name is just that *deconvolution* layers act in the "opposite" way of *convolutions* in terms of tensor shapes.

2.1.5 FCN-32, FCN-16, FCN-8

Thanks to the *Transposed Convolution* and the *Unpooling* layers we now can create contracting structures that extract high-level features followed by expanding sections that upsample the feature map to the original image size: this is exactly what has been proposed by Long et al. [75] contextually to the introduction of the concept of *Fully Convolutional Network* at CVPR 2015.

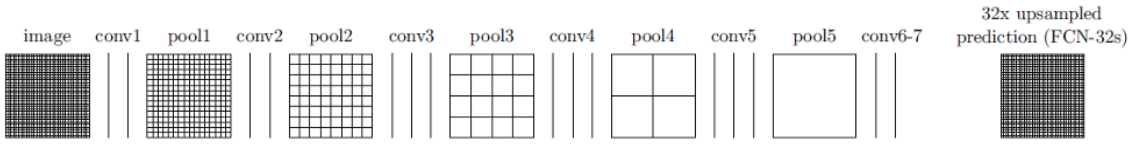

Figure 2.6: FCN-32 [75]

Their network had a traditional contracting section made of subsequent convolutions and maxpoolings that was able to extract high-level features from the original

image and created an high resolution segmentation output, originally using a bilinear interpolation upsampling layer at the end of the chain. It's easy to notice that in such a network, called FCN-32 and depicted in 2.6, spatial information would be gradually lost going down the chain as maxpoolings produce coarser low-resolution feature maps, the upsampled versions fo these maps would necessarily miss most of the original granularity of the inputs. To fight this, Long et al. produced two other networks, FCN-16 and FCN-8, which combined in summation, before the final upsampling stage, partially upsampled versions of the feature maps in the lower stages with the output of pooling layers up the chain: this helped the model recover some of the localization information that was lost in the last stages while mantaining high level comprehension of the context.

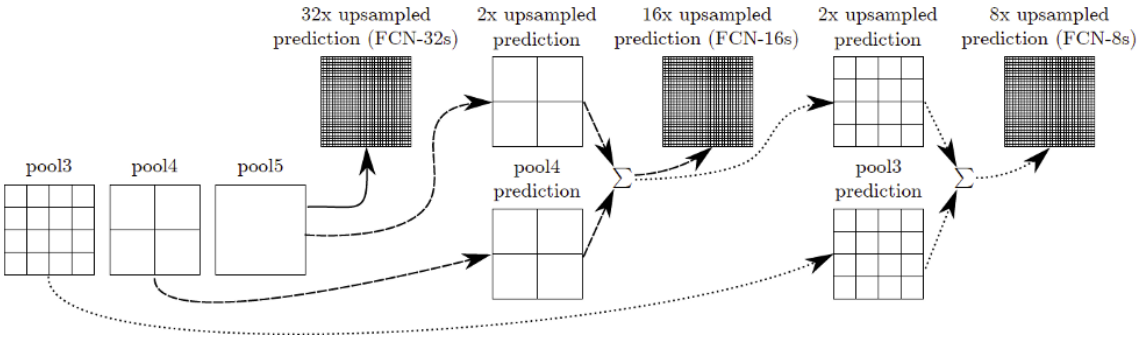

Figure 2.7: FCN-16 and FCN-8 [75]

As seen in figure 2.8, the additional information helps the network to retrieve some of the spatial information that was inevitably discarded during the contracting phases.

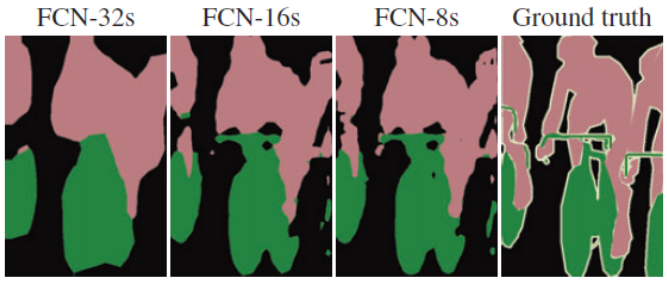

Figure 2.8: FCN-32 outputs compared to FCN-16 and FCN-8 [75]

2.1.6 U-NET

The next step in semantic segmentation using Fully Convolutional Neural Networks is represented by U-NET by Ronneberger et al. [99], depicted in figure 2.9.

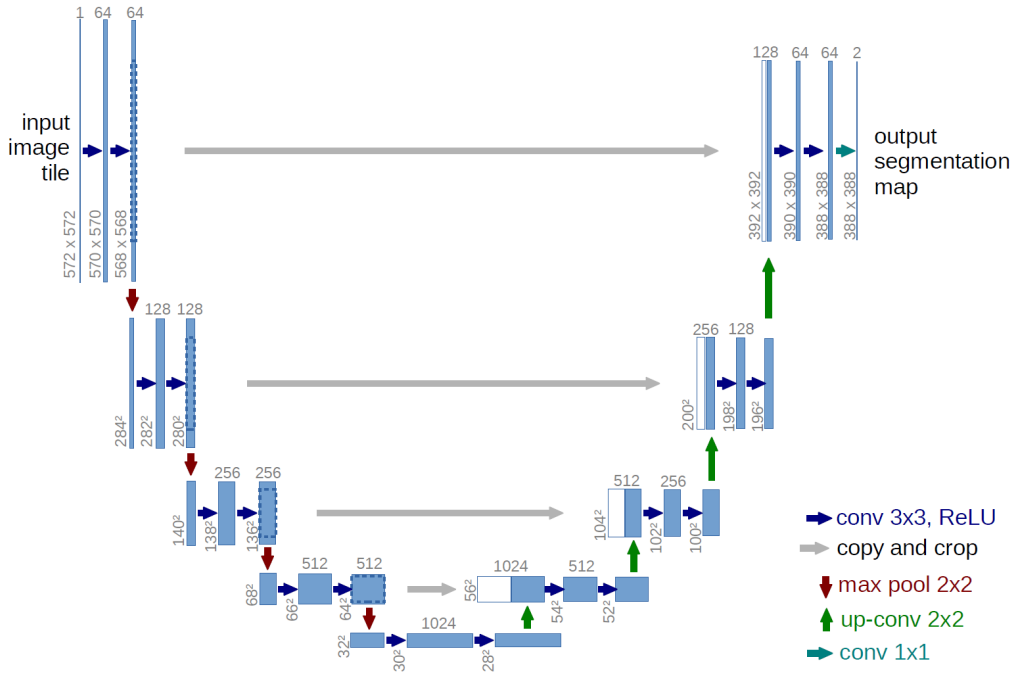

Figure 2.9: U-NET [99]

U-NET, at the first look, appears very different from FCN-32, FCN-16 and FCN-8 but on a closer inspection, it clearly mantains and improves most of the original insight:

- the network is no longer a long contracting path with a single upsampling stage at the end - like the one in FCN-32 - but presents two symmetrical contracting and expanding stages with the same number of convolution and pooling/unpooling stages. Each stage is made of two 3x3 convolutional filters and a 2x2 maxpooling layer.

- the upscaling stages are now *Transposed Convolutions*, meaning that the upscaling parameters can be learned instead of relying on deterministic bilinear interpolation

- the idea of combining deep features with the ouptuts of previous poolings to retrieve spatial information is preserved but implemented in a very different way: while FCN-16 and FCN-8 use summation to combine upscaled outputs to pooling layers, in U-NET previous pooling layers are *concatenated* to the up-convolution output tensors (if the convolution layers don't use "SAME" padding, meaning that the image size across the convolution filters is not preseserved, the previous layers are *cropped* to the correct size before being concatenated).

This kind of network, created for biomedical image segmentation, received spectacularly accurate results in the ISBI 2015 cell tracking challenge, achieving IoU scores of 0.92 while the second best could only score 0.83, establishing itself as the new state of the art in biomedical image segmentation.

2.2 From 2D to 3D Segmentation

In the last section we have introduced some of the methods for bidimensional CNN segmentation: models produce a segmented output starting from a standard 2D image. There are problems, like the segmentation task we will review in the next sections, that require us to extend the segmentation framework from 2D images to volumetric data.
Tumour segmentation in MRI brain volumetric data can be the case: in this scenario the MRI setup is able to provide a stack of co-registered scans of the brain which can be interpreted volumetrically, what we want the CNN model to do is provide us a volumetric segmentation of eventual anomalies in the tissue. Another volumetric segmentation scenario -which happens to be the final application field of this work- is the volumetric segmentation of structures in microscopic data: in particular, the final goal of my work is to use volumetric segmentation techniques to segment cellular structures in Two Photon Fluorescence Microscopy and Light Sheet Fluorescence Microscopy volumetric data.
There are mainly two kinds of paths for tackling these problems, one of them relies on the 2D Convolutional Neural Network segmentation framework explored in the last section while the other path is to find a native 3D formulation that conforms to the volumetric nature of the data and need other kinds of convolutional layers to

be introduced.

2.2.1 2.5D Approaches

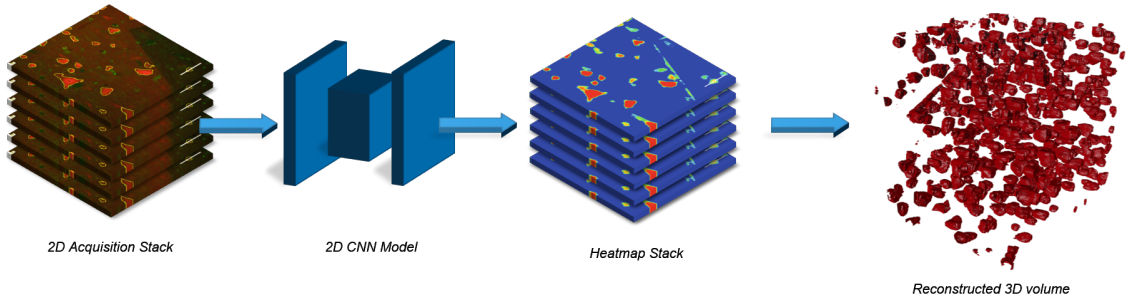

Figure 2.10: 2.5D Approach

A possible way to face the task of volumetric data segmentation is to trace it back to a 2D segmentation task. Assuming that the input data comes in the form of a sequence of registered axial slices of the volume of which we know the spatial resolution characterization in the three axes, we can ideally map every every point of the acquisition data to a point in the 3D space. By serially feeding bidimensional "slices" of the 3D volume to a specialized 2D CNN model one could create a stack of probability images where each pixel value correlates to a class membership probability, effectively building up a cloud of planarly aligned 3D probability points. Using isosurface search algorithms like *Marching Cubes* [76] [70] 3D isoprobability surfaces can be interpolated from this point cloud, giving us a 3D representation of the segmented objects. Figure 2.10 illustrates this kind of workflow: stacks of images from the volumetric input are fed through a 2D CNN model and the resulting heatmaps are combined to form a 3D volume, from this stack of probabilistic representations a surface recognition algorithm can be used to reconstruct 3D meshes corresponding to a given probability threshold. This kind of approach could be seen as a naive interpretation of the problem but has many positive perks: the prediction model does not require more than one slice at the time to make predictions -this could be useful in scenarios where the images are produced and elaborated in real time- and training only involves annotating 2D patches which can be obtained, assuming that the features are sufficiently homogeneous across the volume, from a small part of

the original data.

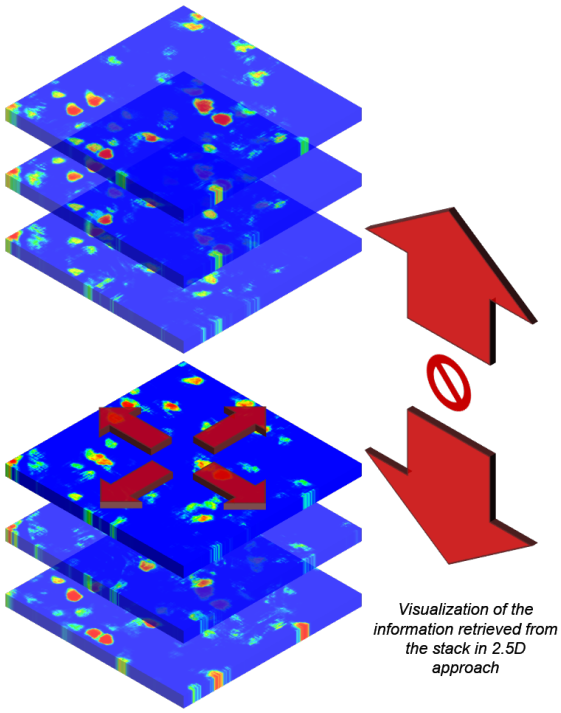

Figure 2.11: 2.5D Approach: Limitations

The main limitation of 2.5D approaches is that, during the slice segmentation phase, the model cannot benefit from the high correlation of visual features between subsequent images of the same stack as every slice is processed separately, meaning that the model can only use 2D contextual information for every single slice while it totally ignores spatial information along the orthogonal dimension wrt the image plane. This kind of limitations become particularly critical when the visual context is not simple enough that the 2D CNN algorithm can accurately segment sections of the objects of interest in every slice: in many cases marginal slices of an object are too small to be effectively distinguished from the 2D context, resulting in inaccurate volumetric reconstructions which appear to be missing marginal details. In figure 2.12 we can see an example of this: it represents a mesh obtained using a surface recognition algorithm on the predictions of a 2.5D model recognizing neuron somas in Two Photon Fluorescence Microscopy volumetric data, the top slices presented too small visual features to be accurately segmented, resulting in inaccurate volumetric

Figure 2.12: 2.5D Approach: "Cropping" Artifacts

estimation of the object.

2.2.2 3D Convolutions

We've seen that 2.5D approaches inherently fail to leverage voxel context from adjacent slices: a way to make use of the discarded 3D information is to redesign the CNN model using 3D Convolutional layers. 3D Convolution is the natural extension of the 2D discrete convolution we've seen in section 1.5.1 and don't introduce any major conceptual difference. In this case both the kernel and the input image are 3D tensors instead of 2D matrices, we can write a 3D version of equation 1.28 just by using the correct summation indices

$$S(i,j,k) = (W * X)(i,j,k) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} X(i-m, j-n, k-l) W(m,n,l) \tag{2.11}$$

or, if we wish, we can switch for the more practical *cross-correlation* operation that can be easily defined as

$$\begin{aligned} S_{\text{cross-correlation}}(i,j,k) = (W * X)(i,j,k) = \\ = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} X(i+m, j+n, k+l) W(m,n,l) \end{aligned} \tag{2.12}$$

Along with the *3D Convolution*, 3D versions of all the other layers like *Pooling* layers and *Transposed Convolutions* can be defined, giving us the building blocks to make entirely three-dimensional CNN architectures.

2.2.3 3D U-NET

One of the most notable examples of 3D Fully Convolutional Neural Networks is the 3D equivalent of U-NET. 3D U-NET by Cicek et al. [133] was presented at MICCAI 2016 as a novel semantic segmentation architecture that could take advantage of 3D space context, as seen in figure 2.13, its structure is very resemblant of U-NET (figure 2.9) with the obvious difference that every layer has been replaced by its 3D equivalent.

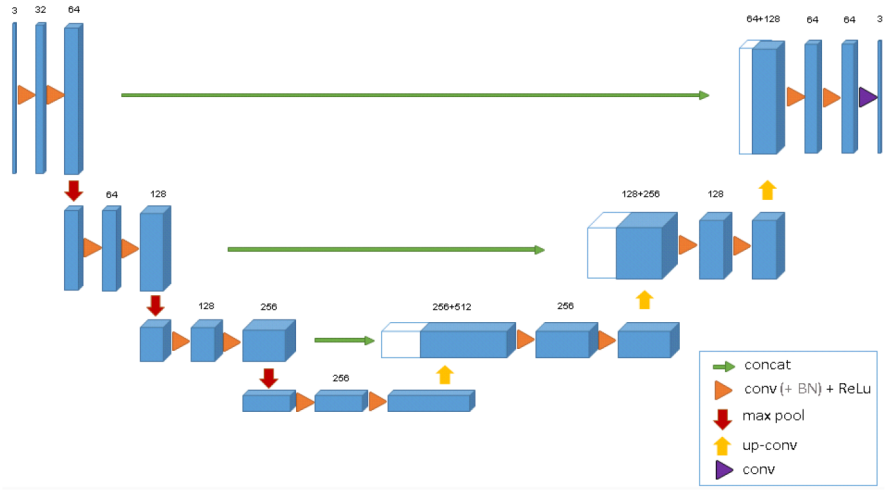

Figure 2.13: 3D U-NET [133]

The 3D U-NET paper received more than 600 citations and is surely establishing a reference point for many CNN segmentation works.This is the foundation for the 3D CNN models used in this work, which closely match this structure and the working intuitions behind it.

2.2.4 2D CNN vs 3D CNN: General Considerations

Passing from 2D CNN models to 3D CNNs can potentially be beneficial in segmentation task if we consider that 3D CNNs can exploit 3D spatial context that

2D CNNs inherently discard but training a 3D model poses a series of new and nontrivial challenges that are related to the very nature of the task.

2D and 3D segmentation, although being naturally linked in a conceptual way, are two very different tasks.

Let's say we want to build a 2D semantic segmentation model which can distinguish between a foreground and a background class based on a optimized version of the *sliding windows* approach, as described in section 2.1.1. the model we're trying to train is functionally a monodimensional classifier that predicts a central pixel class membership (the pixel is either foreground or background) with a patch of pixels - let's say 64×64 - as receptive field. The model implicitly maps a function f from a space $\mathbb{R}^{64\times64} = \mathbb{R}^{4096}$ to the space of possible outcomes, which is $\mathbb{R}$ in if the model predicts a probabilistic output instead of a simple label

$$f : \mathbb{R}^{64\times64} = \mathbb{R}^{4096} \rightarrow \mathbb{R} \tag{2.13}$$

The problem's scales change when, instead of predicting just one label from a single patch like in the sliding window approach, we want to predict probabilistic segmentation labels with the same size of the input image. Now we're not mapping to the real axis, but to a space with same cardinality as the original one

$$f : \mathbb{R}^{64\times64} = \mathbb{R}^{4096} \rightarrow \mathbb{R}^{4096} \tag{2.14}$$

When 3D convolutions are taken into account the cardinality of the involved spaces inflates rapidly: now we don't have just 64×64 input pixels but we want to work with volumetric patches of shape $64 \times 64 \times 64$ so that the new model has to learn a mapping function such that

$$f : \mathbb{R}^{64\times64\times64} = \mathbb{R}^{262144} \rightarrow \mathbb{R}^{262144} \tag{2.15}$$

Massively increased dimensionalities mean that by naively passing from a 2DCNN to a 3DCNN model we would certainly occur in the so-called *curse of dimensionality*: a 3D CNN model cannot work with the same number of examples as a 2D one as geometrically increasing numbers of examples are needed in order to make an accurate representation of a probability density in an high dimensional space.

Most of the time one does not have access to such large amounts of additional data and the only way is to artificially increase the number of samples by data

augmentation: data augmentation stages have to be carefully crafted in order to obtain 3D CNN model convergences.

The other thing to take into consideration is the increased computational complexity of the model itself: both training and evaluating phases are expected to take significantly more resources when compared to the 2D case.

With limited data availability and limited computational resources it's possible that the potential advantages of 3D CNNs in segmentation performance wouldn't be sufficiently backed up by on-field experimental results in the context of this work other than in limited and circumscribed cases.

Part II

Problem Framing and Methods

Chapter 3

Motivation and Quantitative Study of the Brain

The human brain, then, is the most complicated organization of matter that we know.

Isaac Asimov

Understanding the human brain is one of the greatest challenges that the 21st century science will face: getting a grasp on the inner workings of our central nervous system at different scales can help us understand ourselves while making leaps in medical treatments for brain disease and developing new ICT. Current-day neurosciences are extremely developed and prolific but the unsystematic way knowledge of the brain is developed makes it suffer from major fragmentation. It's in an effort to create a common framework for coordinating and powering brain studies through multidisciplinar interaction that large scale international programs, like the Human Brain Project or the BRAIN Initiative are born. Such large-scale projects aim to develop an entirely new paradigm in neuroscience and brain studies by producing large scale and extensive knowledge with the coordinated effort of multiple research groups working on many different aspects: from biological research, to physical study of imaging solutions to computer simulations of models based on experimental data.

To give a sense of how extensive these frameworks are let's take a look at the European international effort in quantitative brain study, the Human Brain Project (HBP) aims to put in place an ICT-based scientific infrastructure organized in four different main goals, each serving as a catalyst for future research: a *Data* goal that

aims to create extensive brain atlases, a *Theory* goal that wants to identify what mathematical principles are at the base of the relationships between different levels of the brain organization as well as a theoretical study of the brain's ability to represent store information, an *ICT Platforms* goal to provide innovative ICT systems for neuroscience research and, lastly, an *Applications* goal to develop prototype technologies based on the brain study results.

3.1 Human Brain Imaging at LENS

The European Laboratory for Non-Linear Spectroscopy (LENS) in Florence, the Biophotonics group, led by Francesco Saverio Pavone is involved in one of the HBP subprojects and currently develops techinques for fluorescence microscopy brain imaging. Having conducted studies on Fluorescence Microscopy imaging of whole mouse brains, using both Two Photon Microscopy (TPFM) and Light Sheet Micrsocopy (LSFM), the group is now applying the same approaches to large-scale imaging of human brain tissue. For this kind of imaging to serve as the basis for the creation of complete atlases of the human brain at cytoarchitectural level and to be used a solid foundation for other higher-level quantitative studies to build onto, an interpretative layer has to be added to the "raw" images coming from the microscopy apparatus. For models to be built onto these atlases, high level information has to be extracted from large raw volumetric acquisitions which, from a computational standpoint, are just large matrices of values with no additional information on *what* the image is representing and *where* the objects are. For this kind of information to be available for computational query, the images need to be segmented, in an automated way as manual image segmentation is simply not feasible in terms of human work-hours.

In 2018 the Biophotonics group at LENS managed to achieve automated segmentation of neuron somas in TPFM using Convolutional Neural Networks [80]: this result made it possible to sequentially process large bidimensional optical sections of brain tissue and stack the results to create 3D reconstructions of the objects. This kind of solution takes with it some inevitable artifacts, some of which have already been covered in Chapter 2. The avoidance of those intrinsic artifacts, along with the suggestion of a path to improve segmentation performances in the next future, is the main motivation for this work and for my effort to reformulate LENS's

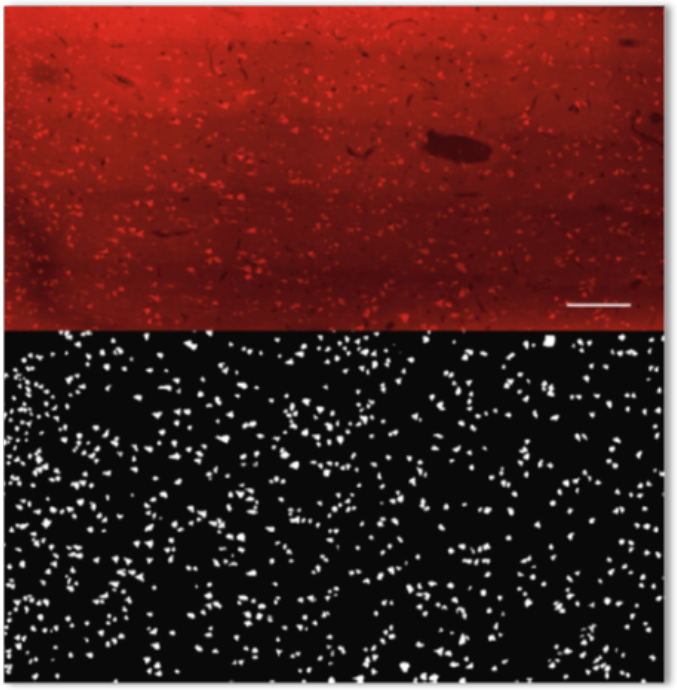

Figure 3.1: Raw and Segmented Fluorescence Microscopy Data [80]

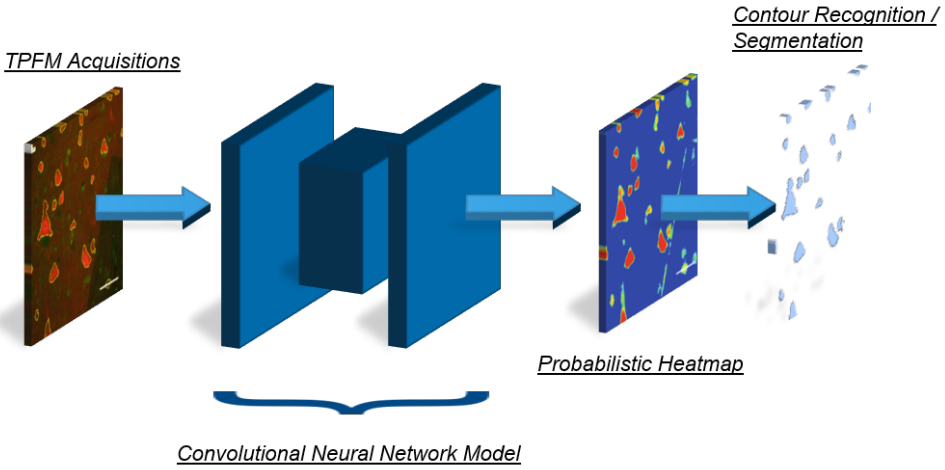

Figure 3.2: 2D-CNN Segmentation Model

CNN neuron segmentation problem into a 3D native format. The switch from traditional CNNs to 3D CNNs, from processing images to processing volumes poses

a series of new theoretical and technical challenges that need to be addressed and solved and will be subject of future research. This work's primary aim is to provide proof of applicability of 3D Convolutional Neural Network methods to the segmentation problem and not to offer a drop-in replacement solution: as a matter of fact the increased resource requirements, in terms of memory, GPU and CPU computing power and human labeled data are likely to require a full redesign of the acquisition and elaboration process in terms of pipeline and needed hardware. If, on one side, this kind of switch can potentially not be immediately implemented, on the other posing a research question in a new direction can potentially bring a change of perspective on the problem and hopefully help to overcome the limitations of the current approach.

3.2 Program for the Next Chapters

This work stems from the necessity of finding new approaches to the neuron segmentation problem at LENS. Because of this the choice of this particular route of action in Deep Learning techniques followed an extended phase of study of both the biological subject and the physics of the involved imaging methods. Understanding the subject is a necessary step to think an adequate solution, so we wanted this work to not only be focused on Machine Learning and Deep Learning aspects of the proposed models but also to reflect the processes that lead to the choice of a particular form of Machine Learning model.

The next three chapters will offer a **biological**, an **imaging/physical** and a **machine learning** framing for the segmentation problem we introduced and partially discussed in chapter 2. If the choice of multiple perspectives that, on a first glance, do not significantly overlap seems odd to the reader, it is not fully unorthodox from a modern neuroscientific point of view that a specialist -being him a physicist, a biologist, a chemist, a mathematician....- has to acquire knowledge out ouf his initial expertise field in order to come up with new solutions for problems in such a multidisciplinar environment as the one created by large programs like HBP. In a sense, we wanted this work to reflect, at least partly, the many-discipline collaborative environment it has been conceived into and hopefullythe next three chapters will reflect that plurality of views and perspectives.

Chapter 4

Fluorescence Microscopy

It was certainly a curious sight to see the tube instantaneously lighted up when plunged into the invisible rays: it was literally ***darkness visible****.*
Altogether the phenomenon had something of an unearthly appearance

George Gabriel Stokes
On The Change of Refrangibility of Light, 1852

This chapter wants to expand onto the physics and methodology behind the acquisition process of the volumetric imaging stacks of cortical tissue. Two technologies were involved in the generation of this data, both relying on different aspects of the phoenomena: Light Sheet Fluorescence Microscopy and Two Photon Fluorescence Microscopy. In the next sections we will touch upon the basics of Fluorescence Microscopy and its most relevant principles.

Fluorescence is the emission of light caused by radiative transitions from excited singlet states of a molecule to the singlet ground state. Excited states are reached via the absorption of a photon with wavelength generally shorter than the emitted one: the energy loss is due to vibrational relaxation of the molecule in the excited state, this shift towards higher wavelengths is called *Stokes Shift* after Sir George Gabriel Stokes, who first suggested an explanation for the phoenomena his welll-known 1852 paper *On the Change of Refrangibility of Light* [114].

Records of fluorescence date back to late sixteenth century when Bernardino de Sahagùn, Franciscan missionary in Mexico, described what at the time were unexplicable luminescences of an infusion of *lignum nefriticum*, scientifically *Eysenhardtia polystachya*, a native plant of Central America: what Sahagùn observed were actually the fluorescence properties of *formononetin*, an oxidation product of one of the flavonoids found in the wood. Three centuries later Sir Gabriel Stokes documented the properties of uranium glass and fluorite (hence the *fluorescency* term) that were observed to emit visible light when exposed to ultraviolet wavelength, isolating for the first time the real mechanism of fluorescence.

Fluorescence phoenomena serve as base for *Fluorescence Microscopy*: the main idea behind the technique is that, instead of illuminating the sample with visible light and observing in the same spectrum, the sample (generally organic tissue) is exposed to specific wavelengths that can trigger photon absorption and successive fluorescence in specific markers, called *fluorophores*. The variety of available fluorophores makes it possible to selectively study different aspects of the biological sample depending on their chemical and biochemical properties. Biological samples can be labelled with fluorescent stains which can be targeted to specific biomolecules exploiting immunological process by binding antibodies (proteins belonging to the *globulin* group, synthetized by B-lymphocytes) or antigens to fluorophores: this technique is commonly known as *immunofluorescence*.[1] Current knowledge of genetic manipulation techniques is also used to modify DNA code to make proteins of interest fluorescent: this allows to track those proteins even in live samples.

In the next paragraphs we try to give a minimal overview of fluorescence phoenomenon in its generalities and shed some light on some of the applications that made this work possible.

[1]Immunofluorescence techniques are generally of two kinds: a direct, simpler type uses specific antibodies obtained by immunizing an animal against an antigene we're interested in knowing the spatial distribution, the other indirect process sees the antigene reacting with an unmarked primary antibody, the antibody-antigene complex is then made visible when the bonding with a secondary, previously marked, antibody. Many secondary antibodies can possibly bond to the primary antibody-antigene complex, amplifiyng the fluorescence signal.

4.1 The Jablonski Diagram

Jablonski Diagrams, proposed for the first time by Polish physicist Alexandre Jablonski in 1933 [53], are a way of visualizing electronic states of a molecule and the transitions between them. In Fig 4.1 we see a typical diagram depicting levels and transitions of a fluorescent molecule, electronic states are represented with thick lines, whereas coordinated vibrational levels are depicted as thinner lines: in this case singlet ground, first and second electronic states labeled respectively as S_0, S_1 and S_2, successive lines represent coordinated vibrational levels.

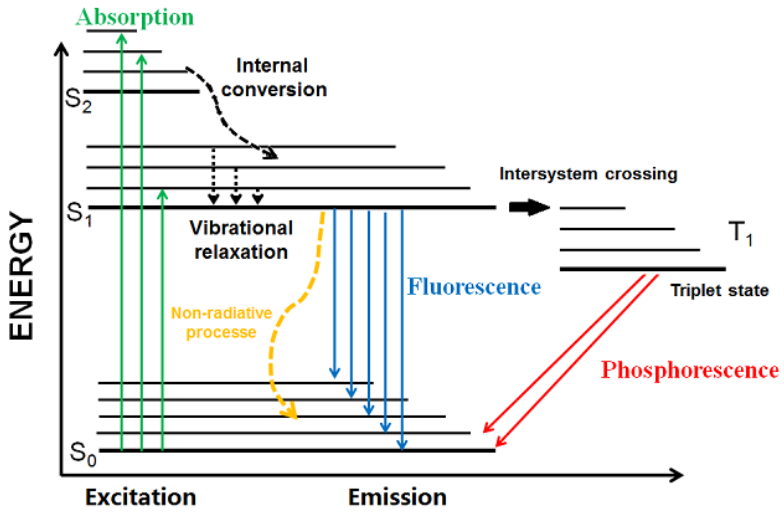

Figure 4.1: Jablonski Diagram

Arrows indicate the possible transitions between levels: radiative transitions are generally represented by straight lines whereas curved ones specify nonradiative processes.

Looking at Figure 4.1 we can see that excited states can be reached from the ground state via the **absorption** of one photon, effectively promoting an electron into a new orbital with higher energy. Absorbption processes usually happen in a few femtoseconds, not allowing the positional adjustment to equilibrium of the much heavier nuclei, the so changed charged distribution results in net forces on the nuclei themselves causing vibrational mode excitation and successive dissipation in the surrounding medium (indicated by short dotted vertical lines) until a new equilibrium position is reached, these processes happen in very short timescales between

10^{-14} to 10^{-11} seconds. From here S_0 and correlated vibrational states are reachable via spin-allowed photon emissions, such type of emission is named **fluorescence**: this process happens in time spans ranging from 10^{-9} to 10^{-6} seconds.

Fluorescence is not the only possible relaxation process and not even the only optically-observable one. If there's significant overlapping between different electronic and vibrational levels, crossover of two states with same multiplicity is possible, leading to subsequent vibrational relaxation: this whole mechanism is known as **internal conversion** and happens in the same timescale as normal vibrational relaxations. Another concurrent pathway is **intersystem crossing**: in this case we have crossover between electronic states with different multiplicity, passing from an excited singlet state to an excited triplet state, this is a *forbidden* transition that's not allowed by regular selection rules and thus happens in a much wider timescale, taking anywhere from 10^{-8} to 10^{-3} seconds. After intersystem crossing further deactivation may happen through **phosphorescence**. The decay of a triplet state back into a singlet state is another forbidden transition due to different spin multiplicity and the requirement of angular momentum conservation, however spin-orbit coupling is observed to explain a relaxation of this last restriction letting phosophorescence ultimately possible: timescales for this last process are extremely large, spanning from 10^{-3} to 100 second orders.

4.2 Fluorescence Quantum Yield and Fluorescence Lifetime

We've discussed how an excited electron can undergo radiative and non-radiative relaxation paths without worrying about introducing some kind of quantification for what are the relative ratios for the various processes: we shall now introduce the **fluorescence quantum yield** as a way of keeping track of this kind of information. **Fluorescence Quantum Yield** is defined as the ratio of the number of emitted photons to the number of absorbed ones

$$Q = \frac{n_{\text{emitted}}}{n_{\text{absorbed}}} \tag{4.1}$$

Helping ourselves with a simplified version of the Jablonski Diagram in which we

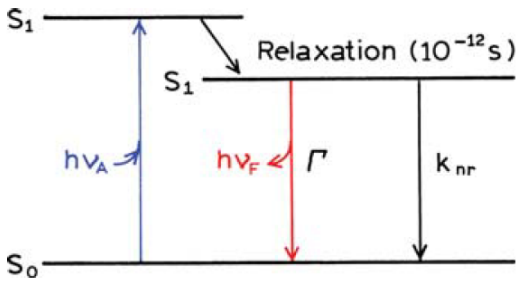

Figure 4.2: Simplified Jablonski Diagram [67]

focus only on processes responsible for return in the ground S_0 state we can express the above quantity in terms of rate constants: in particular we're interested in the emissive rate Γ and the rate of non-radiative decay k_{nr} [2]

$$Q = \frac{\Gamma}{\Gamma + k_{nr}} \tag{4.2}$$

Lifetime of the excited state can also be evaluated using emissive and non-radiative rates:

$$\tau = \frac{1}{\Gamma + k_{nr}} \tag{4.3}$$

In case of absence of non-radiative processes the *fluorescence lifetime* is named **natural lifetime** and can simply be expressed as

$$\tau_n = \frac{1}{\Gamma} \tag{4.4}$$

combining 4.4 with 4.3 and 4.2 we can then write *natural lifetime* in terms of *fluorescence quantum yields* and *lifetimes*

$$\tau_n = \frac{\tau}{Q} \tag{4.5}$$

Finding a way of express Γ in terms of known quantities would help us complete our picutre: as a matter of fact Γ can be calculated from the fluorophore's emission

[2] in k_{nr} we've summarized all non-radiative processes: if we wish to keep track of single non-radiative rates we might as well use the notation $k_{nr} = \sum k_{nr_i}$

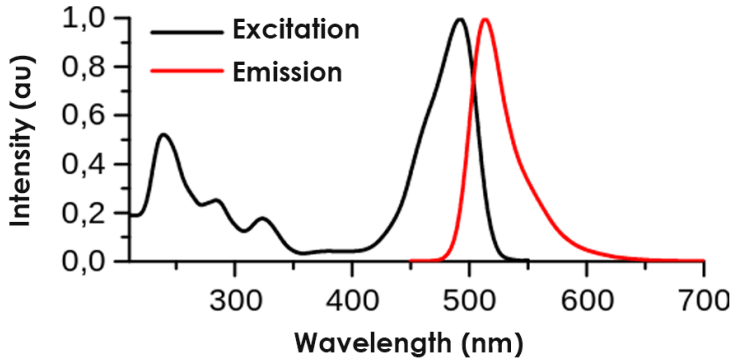

Figure 4.3: Absorbption and Emission spectra of fluorescein $C_{20}H_{12}O_5$

and absorbtion spectra, which can be directly measured, like the ones depicted in figure 4.3 for the fluorescein $C_{20}H_{12}O_5$ molecule. Calculations involves nontrivial integrals that can be numerically computed:

$$\Gamma \approx 2.88 \times 10^{-9} n^2 \frac{\int F(\bar{\nu}) d\bar{\nu}}{\int \frac{F(\bar{\nu}) d\bar{\nu}}{\bar{\nu}^3}} \int \frac{\epsilon(\bar{\nu})}{\bar{\nu}} d\bar{\nu} = 2.88 \times 10^{-9} n^2 < \bar{\nu}^{-3} >^{-1} \int \frac{\epsilon(\bar{\nu})}{\bar{\nu}} d\bar{\nu} \quad (4.6)$$

where F and ϵ are respectevely the emission and absorption spectra and n is the refractive index of the medium: for fluorescein the obtained lifetime is 4.0 ± 0.1 nanoseconds.

However, the above formula ignores interactions with the solvent and does not account for differences between absorption and emission refraction index. Better results can be obtained accounting for degeneracies in the lower and upper states, g_l and g_u, introducing a $G = g_l/g_u$ multiplicative factor.

4.3 Two-Photon Excitation

Multiphoton excitations are non-linear processes involving the simultaneous absorption of two or more photons in a single event, in particular **two-photon excitation** processes (known as **2PE** opposed to single-photon excitation **1PE**) see the cooperation of two low-energy photons (typically from the same laser source) to cause a higher-energy electronic transition in a fluorescent molecule. Two-photon absorption

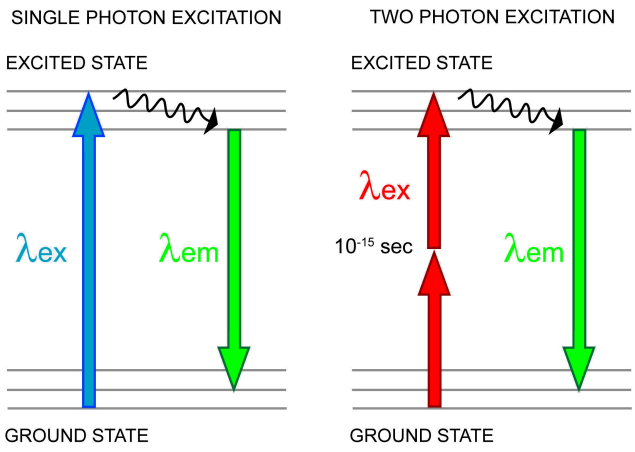

Figure 4.4: Simplified Jablonski Diagram Two-Photon Excitation

rate depends on the second power of the light intensity, this effect can be exploited to increase localization simply by focusing the excitation beam.

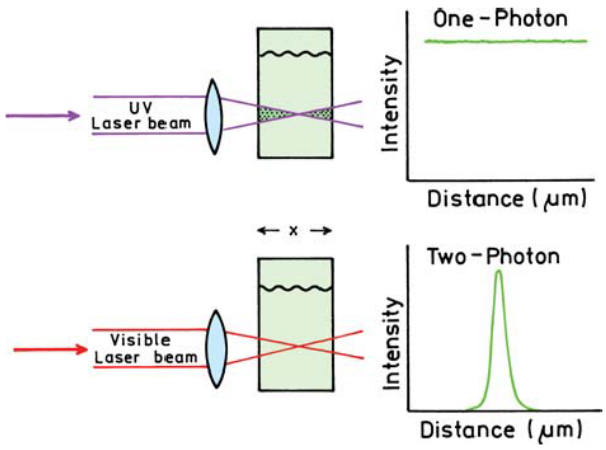

Figure 4.5: Two Photon Excitation localization effect [67]

Let's take the case of a focused beam passing inside a photoluminescent medium: figure 4.6 quickly summarizes the cases of single-photon and two-photon excitations, in the *1PE* case focusing the beam doesn't change the total amount of passing light at a certain position x keeping absorption constant on all the x planes (assuming neglegible attenuation), when we consider *2PE* instead we can see that absorption

is highest in proximity of the focus and quadratically drops with distance: this is due to the fact that absorption rate depends quadratically on the intensity.

The main consequence of the localization of excitation is an increased contrast and resolution in three-dimensional imaging, other than reduced photobleaching (i.e. the photochemical alteration of fluorophores such that they lose their fluorescence properties) and photodamage (biological damage to the tissues due to electromagnetic interaction) because of the smaller exposed areas, whereas in single-photon microscopy the entire thickness of the sample would have been subject to photobleaching even if data was only collected from the focal plane. Other advantages over $1PE$ techniques are represented by the use lower wavelengths that better penetrate tissue when compared to visible wavelengths used in one-photon microscopy because of reduced scattering and absorption by endogenous chromophores.

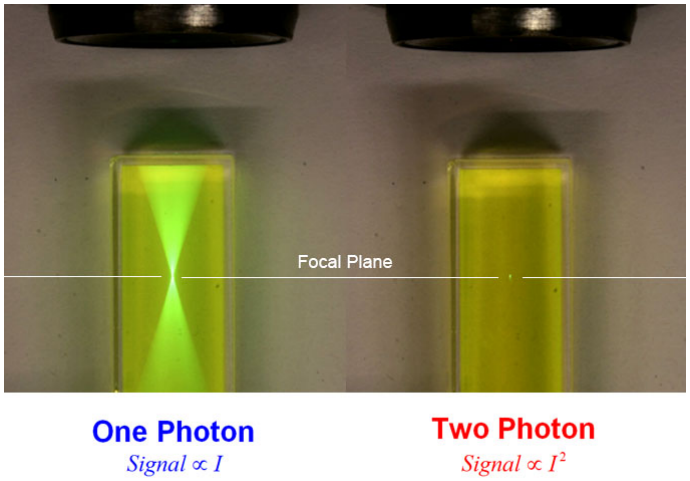

Figure 4.6: Single Photon Excitation vs Two Photon, *Image by Xu Research Group, Cornell University*

As a consequence of the highly localized excitation, contributions due to scattered photons are limited to small focal volumes around the focal points instead of being spatially extended over the sample: this is one of the reasons why, while single-photon microscopy generally needs a detector pinhole to allow only light generated in the spatial region of interest to collected, in two-photon microscopy all photons collected by the objective constitute useful signal.

Because of two-photon excitation small cross sections high intensities are required: these are possible with te use of Ti:sapphire mode-locked lasers producing trains of pulses, illuminating small portions of tissue with efficiencies increasing as the inverse of pulse durations, typically in the order of 100 fs.

4.3.1 Comparing the numbers

Willing to do a more objective comparison between 1PE and 2PE techniques let's consider 1PE and 2PE cross sections. Single-photon excitation cross sections are measured in cm^2 and offer an intutive geometrical interpretation, they typically range from $10^{-17}cm^2$ to $10^{-15}cm^2$ and can be seem as effective interaction square areas of 0.3Å to 3Å side, compatible with fluorophore physical sizes. On the other hand, 2PE cross sections are measured in *GM* units (Goppert-Mayer, from the name of Maria Goppert-Mayer, who elaborated the two-photon absorption process theory), where $1GM = 10^{-50}cm^4s/photon$: in this case the geometrical interpretation is not as trivial.
The use of this unit can be understood by looking at the single-photon and double-photon cases: in the 1PE case we can express the *number of absorbed photons per second* as

$$N_{1PE}[photons/s] = \sigma[cm^2]I[photon/cm^2s] \tag{4.7}$$

where σ is the 1PE cross section and I is the photon intensity. If we wish to write down the same quantity for 2PE we shall take in consideration the quadratic nature of the process: in this case the number of absorptions per second will be proportional to the square of the intensity

$$N_{2PE}[photons/s] = \delta I^2[photon/cm^2s]^2 \tag{4.8}$$

where we used δ to indicate the 2PE cross section. Comparing the two quantities we can easily get a grasp of what's the correct interpretation of *GM* units.

To better tackle the problem of comparing the two processes we can imagine 2PE cross sections having both a "spatial" and a "temporal" component since both photons are required to interact with the fluorophore inside a small time window,

so that 2PE cross sections can be seen[3] as the product of two areas (one for each photon) and an interaction time: if we assume the spatial component to be comparable with the 1PE one, given that single-photon excitation would be a "spatial-only" cross section, we can think it to be in the order of $10^{-16}cm^2$. That would leave us, for a 2PE cross section in the order of $1GM$, with a $10^{-18}s/photon$ temporal component which gives us an idea of what the allowed temporal distances between the two interacting photons would be.

On the power requirements side, we can compare fluorescence saturation in the two cases. We start by defining the *fluorescence saturation* situation as the one in which emission rates equate absorption ones. Adopting the inverse of a typical fluorescence lifetime as the emission rate and using an indicative cross section $\sigma = 10^{-16}cm^2$ we can use equation 4.7 to retrieve the 1PE required intensity:

$$\begin{aligned} N_{1PE} &= \frac{1}{\Gamma} = \sigma I_{1PE} \\ I_{PE} = \frac{1}{\Gamma\sigma} &\approx \frac{1}{4ns \cdot \text{photons}^{-1} 10^{-16}cm^2} = 2.5 \cdot 10^{24}\,\text{photons}\,cm^{-2}s^{-1} \end{aligned} \tag{4.9}$$

The amount of power that a $488nm$ continuous wave laser would need to bring a focused diffraction-limited area of $(488)^2nm^2 = 2.4 \cdot 10^{-9}cm^2$ to saturation can be calculated. The energy of a single emitted photon would be

$$E_{SP} = \frac{h\,c}{\lambda} = \frac{6.62 \cdot 10^{-34} J\,s \cdot 2.99 \cdot 10^{10} cm\,s^{-1}}{4.88 \cdot 10^{-5}\,cm} = 4.06 \cdot 10^{-19}\,J\,\text{photons}^{-1} \tag{4.10}$$

so we can express the irradiance or flux density for such a source as

$$\begin{aligned} E_{488nm} &= E_{SP} \cdot I_{PE} \\ = 4.06 \cdot 10^{-19}\,J\,\text{photons}^{-1} \cdot 2.5 \cdot 10^{24}\,\text{photons}\,cm^{-2}\,s^{-1} &= 1.0MW\,cm^{-2} \end{aligned} \tag{4.11}$$

If we multiply the flux density by the needed area we obtain a total needed power of $P_{1PE,sat} = 2.4mW$: less than the power generated by a laser pointer.
Two photon excitation, on the other hand, would require both photons to arrive in

[3]This comparison is to be intended just as a visualization tool and does not describe the actual physics of the process.

a $10^{-18}s$ time window and be closer than 3Å. To obtain the same N_{1PE} number of absorbed photons per second given by an $I_{1PE} = 1MWcm^{-2}$ we would need intensities in the order of $I_{2PE} = 10^6 MWcm^{-2}$, 100000 times the $1PE$ case: these continuous-wave intensities would be comparable to the ones used in laser welding, definitely not suitable for biological-tissue application. The loophole here is that we don't necessarily need continuous power: mode-locked Ti:sapphire lasers can output $10^{-13}s$ pulses at a repetition rate of $100MHz$, staying off 10^5 more times the intervals that they're actually on, while the pulse itself is actually 10^5 times longer than the time window we need for $2PE$ events to be possible. Instead of kilowatt-like consumptions, by using mode-locked lasers, we can reach the needed saturation intensities using just tens of times the power needed for 1PE microscopy with CW sources.

4.4 LighSheet Fluorescence Microscopy (LSFM)

Typical aims for fluorescence microscopy generally include finding ways to reduce photobleaching and phototoxicity, capturing highly dynamic processes in three dimensions, improving image quality and minimizing detection contributions of out-of-focus fluorescence: standard confocal microscopy tends to be generally flawed in these areas due to intrinsic geometric limits.

Being that illumination and detection share the same light path, a common issue with confocal microscopy is the unselective illumination of areas outside the focal plane with consequent photobleaching of non-imaged zones (problem worsened during scanning if lightpaths of different patches significantly overlap) or contributions of said areas in the final image.

Also, scanning of the imaged area is a relatively slow process that's not suitable for capturing significantly dynamic processes where temporal consistency of the various acquisitions forming the image is a strict requirement.

An approach for tackling these problems is represented by the set of strategies under the common name of *LightSheet Fluorescence Microscopy*: the main intuition behind them is the decoupling of illumination and detection paths by routing them perpendicularly. The specimen is illuminated with a *laser light sheet* - a laser beam planarly focused in the focal plane using a cylindrical lens - while fluorescence light

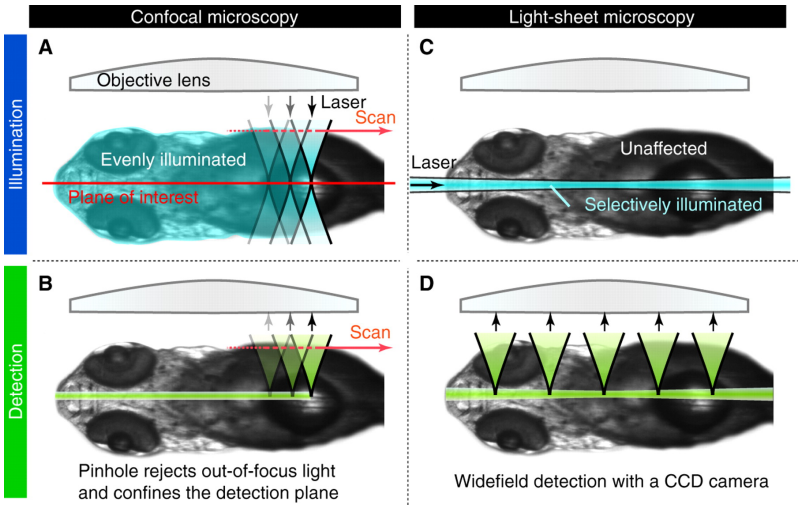

Figure 4.7: Confocal and LightSheet Microscopy Illumination and Detection Paths [48]

from all the illuminated plane is observed on an orthogonal axis using an imaging sensor (usually a CCD or a CMOS camera), as depicted in figure 4.8.
This setup has two main advantages over standard confocal microscopy: only the needed slice of the sample is illuminated, thus greatly reducing photobleaching and phototoxicity effects, and on the other hand, perpendicular observation allows for entire slices to be captured in a single acquisition, greatly reducing detection times.

The orthogonal geometry of lightsheet setups and the relatively quick acquisition times open up various imaging possibilities that are far less feasible using classic confocal microscopy: rotations or translations either of the specimen or the light sheet can be used to achieve multiview microscopy and images from different illumination/detection paths can be merged to obtain symmetrical resolution in 3D sliced imaging. LSFM can be combined with 2PE microscopy with all the advantages described in the previous section.

4.4.1 LSFM-SPIM and LSFM-DSLM

There are multiple geometries for lightsheet microscopes and most of them share the same two working principles to generate a light sheet: in LSFM-SPIM a laser

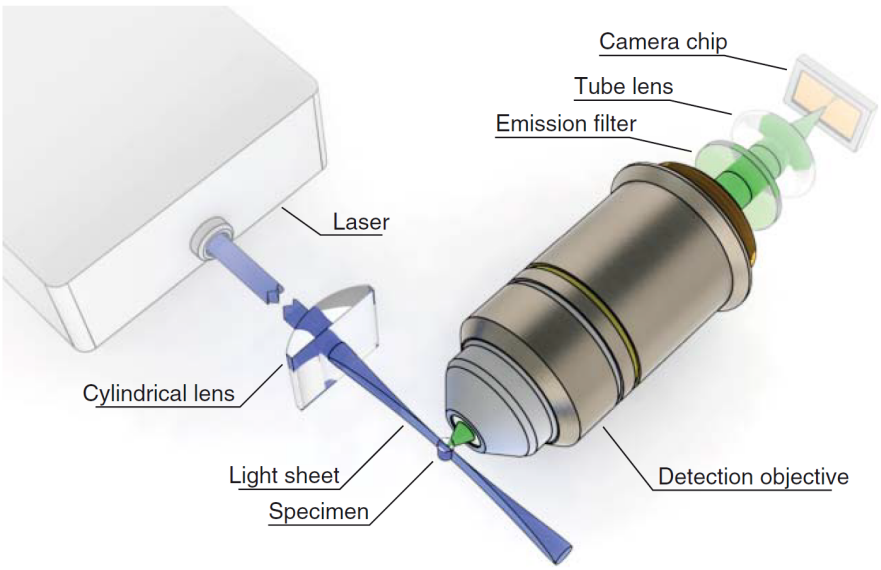

Figure 4.8: LightSheet Setup [103]

beam is expanded and focused using a cylindrical lens to form an actual extended *light sheet* that sections the sample, while in LSFM-DSLM a scanning approach is used: a "virtual" light sheet is generated by scanning in a single direction a focused beam perpendicular to the detection axis.

In figure 4.9 we can see a SPIM and a DSLM setup from a top down and a side perspective.4.9 a) depicts a typical SPIM setup, a laser source of diameter d_b is focused though a cylindrical lens - L_{CV} - and adapted to a diameter d_h using a telescopic system formed by L_{T1} and L_{T2}, the beam, focused on the BFP back focal plane of the illumination objective OL_i wich projects the light on the focal plane of the detection objective OL_d. 4.9 b) is a DLSM setup: here we see no cylindrical lens but a scanning mirror that sweeps a beam across the illumination path, effectively creating a virtual light sheet at the focal plane of the detection objective, the height of the light sheet can be changed by changing the sweep amplitude and the thickness is regulated by varying the diameter of the incoming beam, allowing flexibility to the imaging needs.
Compared to SPIM, DLSM's scanning nature introduces more light, thus higher photobleaching is to be taken into account when planning acquisitions. During the

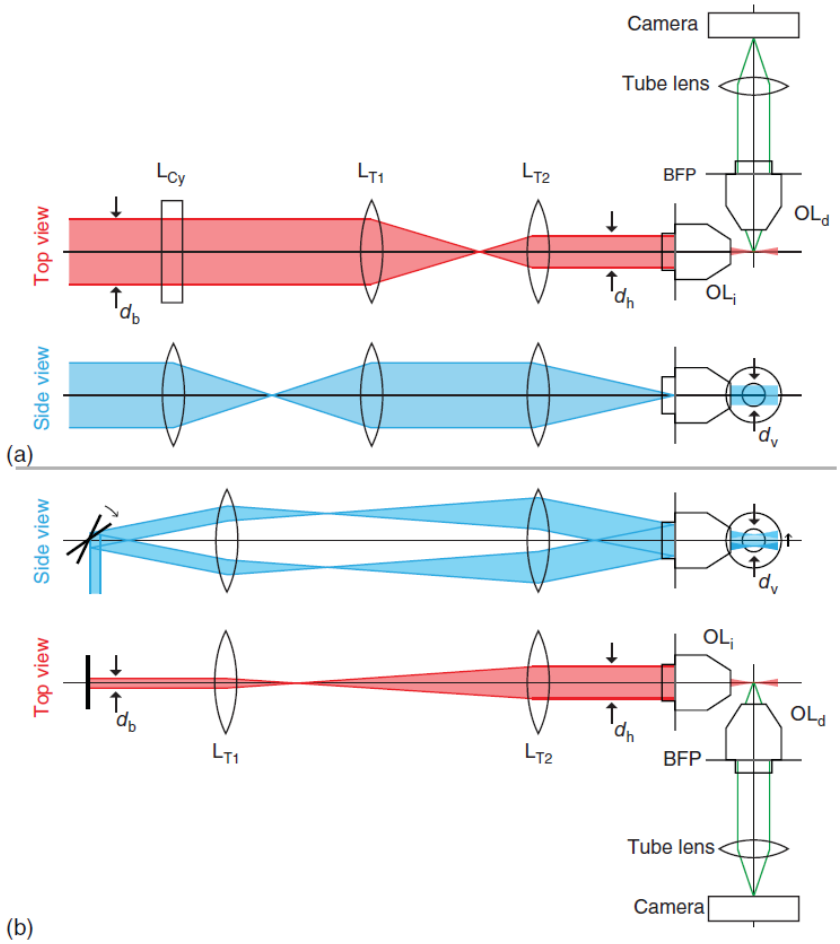

Figure 4.9: SPIM (a) and DSLM (b) light sheet generation methods [64]

scan only a small region of the FOV produces an image, while there are many out-of-focus contributions that would be detrimental to detection accuracy: this is generally taken care of by synchronizing the rolling shutter of the CMOS sensor in such a way that only the well-focused area at a given time contributes to the image, creating a "virtual pinhole" that excludes out-of-focus contributions. When confronting DSLM light sheet to SPIM the former usually has a more uniform intensity profile along its height, however "real" simultaneous plane illumination is preferred for high-speed imaging applications as it avoids motion artifacts.

4.4.2 LSFM Image Properties and Artifacts

LightSheet microscopy image quality are inevitably affected by interaction of light with the sample, producing predictable effects in the final image. Interaction with the sample happens broadly in the forms of **absorption** and **scattering** of the incoming light. Scattering of laser light outside the focal plane produces an increase in thickness of the lightsheet, therefore affecting slice selection capabilities, this effect can be significantly attenuated by choosing an appropriate polarization: scattering in tissue is polarization-dependent so a polarizing filter can be used to maximize scattering in the focal plane (that doesn't impact negatively on the image quality) and minimize out-of-plane scatter events.

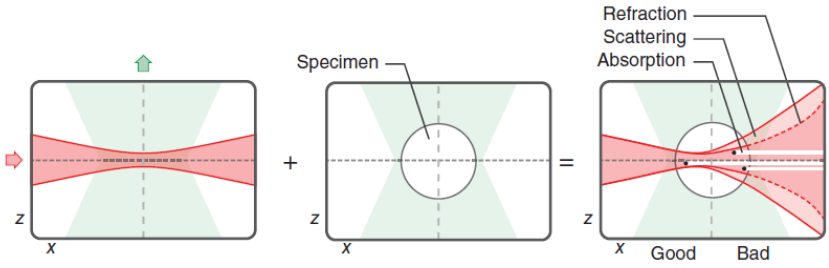

Figure 4.10: Light Sheet Interaction with Tissue [124]

Absorption of excitation photons in a dense medium can be described by the Lambert-Beer law

$$I(x) = I_0 e^{-\mu x} \tag{4.12}$$

where I is the light intensity at a given x position, and μ is an absorption coefficient defined by the medium properties. For an approximately isotropic and uniform environment, we should have an exponential decay of light intensity across the sample, this is generally not the case in biological tissues since they present internal structuring with varying densities and physical properties (good thing, considering that viewing internal structure should be the main interest of microscopical imaging): what is normally observed is a combination of Lambert-Beer exponential (or multi-exponential) attenuation and stripe-like shadowing effects of denser area that are closer to the light source.

One way of mitigating this effect is illuminating the sample from both sides:

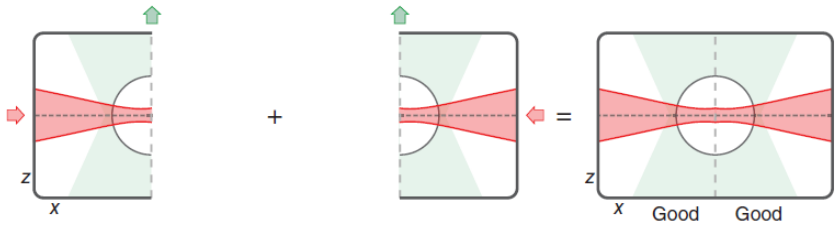

Figure 4.11: Double-Sided Illumination of the Sample [64]

this can be done using two sources, either simultaneously or in an alternated fashion by keeping half of the image (the well-illuminated part) from both takes, as schematically represented in figure 4.11.

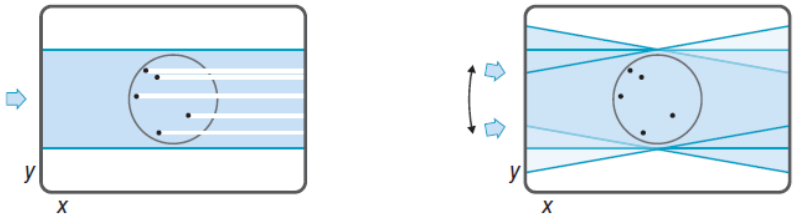

Figure 4.12: Pivoting the LightSheet [64]

Another way of facing the same problem is by *pivoting* the light source: by scanning the light sheet over an extensive angle at kHz frequencies with the use of a resonant mirror during a single exposure of the camera, striping effects can be heavily attenuated as the obstacles are homogeneously illuminated from a range of angles.

4.4.3 MultiView LSFM and LSFM Configurations

LSFM SPIM techniques are naturally affected by scattering and attenuation-related limitations making areas facing illumination appear with higher contrast and signal-to-noise ratios than the less exposed ones. This problem can be faced by acquiring multiple views of the sample in different perspectives: entire z-stacks of the specimen viewed from different angles can be obtained by simply rotating the sample, if acquisition process of a stack is fast enough (or the processes inside the sample slow

enough) to keep different stacks compatible with each other.
Multiple views can then be combined together to obtain global higher-quality information on the sample, an example of that is the possibility represented by perpendicular acquisitions: z-axis resolution is almost always lower than resolution in the detection plane and ,ideally, low resolution in such direction in one dataset can be improved by exploiting higher planar resolutions in the complementary dataset.
To merge multiple-angle z-stacks into higher-quality 3D data, computational approaches for deconvolution of multiple views based on point spread functions recorded from different perspectives have been thoroughly explored, though they suffer from several drawbacks. High precisions in both specimen rotation and rotation axis positioning are needed, but very difficult to achieve, often making images obtained by specimen rotation inherently misaligned: this problem can be confronted from the perspective of fine motor control or by selecting fluorescent fiducial markers inside the specimen itself.
Another problem is the time-related dataset incompatibilities due to biologic processes happening in the period between two acquisitions.
These and other acquisition problems can be dealt with by selecting a particular

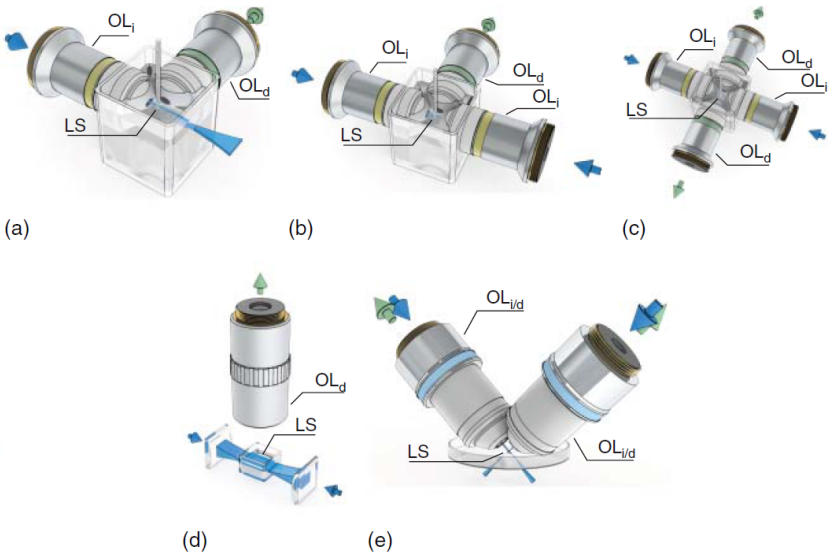

Figure 4.13: SPIM Configurations [64]

LSFM geometry, in figure 4.13 we can see a schematic selection of the most common configurations: a) represents the standard setup with an orthogonal light source and detection path, b) illustrates a three-lens SPIM where a second illumination path generates an additional light sheet to illuminate the sample, c) can be exploited to tackle the problem of simultaneous multi-view imaging [63] with the addition of a separate detection/illumination path, d) ultramicroscope configurations are typically designed around large samples and finally e) dual inverted SPIM (diSPIM) setups provide multiview imaging by combining two different illumination/detection paths in two optical objectives, alternating illumination and detection. The last one is the particular setup used for data acquisition.

4.4.4 Problems Related to Data Handling

LightSheet microscopy, especially in its 3D multiview variants, can generate large volumes of data when high spatial and temporal resolutions are taken into account: data quantities can be orders of magnitude higher than the ones in conventional confocal microscopy, reaching sometimes the same sizes and stream rates of some CERN experiments [96]. Knowing this, real-time elaboration needs to be wisely planned as every step of a pipeline with such massive quantities of raw data is almost assured to be a bottleneck if extra effort is not put into the design phase: everything, from single storage devices writing and reading speeds, to network cards and switches needs to professionally selected ahead to reflect the current (or even near-future) state of the art in the IT panorama.

In figure 4.14 a) we have a comparison of the amount of data generated over 24h by confocal laser microscopy, SPIM using CCD sensors and SPIM using cMOS sensors: the need of difference in approach emerges clearly from a quick size compariso. Figure 4.14 b) offers a comparison of typical hardware differences in designing pipeline for CLSM and SPIM technologies, 4.14 c) table provides examples of suggested pc setups for SPIM data visualization and handling.[4]

[4]specifications are relative to year 2015, when the figure was generated: while 10 Gbit network speeds, storage and memory figures are still reasonable options today, for GPU memory NVIDIA at the present day (2019) offers way more capable graphic processors even at consumer-grade with RTX-2080Ti GPUs having 11 GBs of GDDR6 onboard memory while dual CPU workstation motherboards, in AMD environments, are rapidly getting replaced with EPYC sockets offering up to 64 phyisical cores (128 hyperthreads) with accessible pricetags.

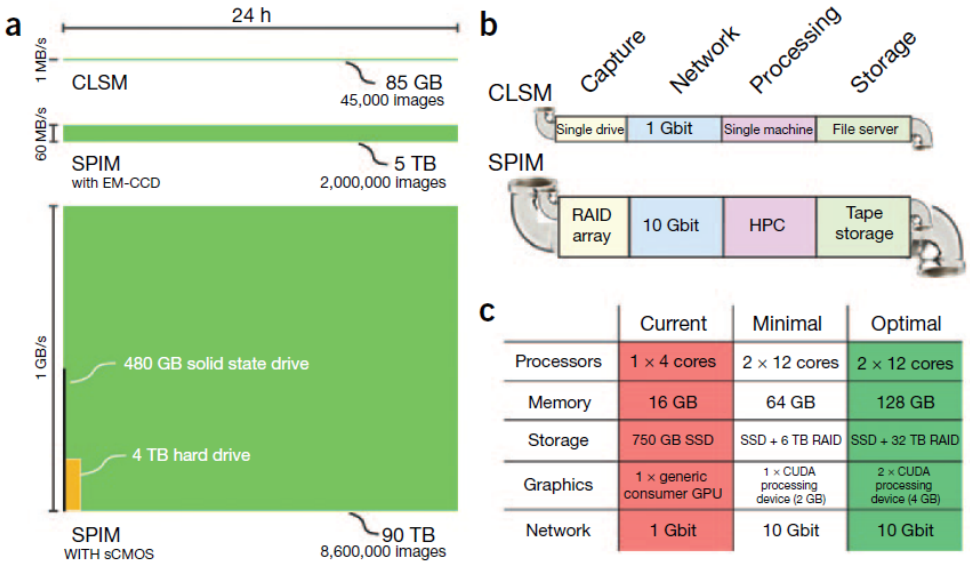

Figure 4.14: Data Generated in SPIM LSFM [96]

Distinctions need to be made between *cold*, *warm* and *hot* data blobs with careful weighting of write speed/access speed/per-gigabyte cost trade-offs, selecting the appropriate storage mean that can ideally range from long-term tape back-up for cold data to NVMe-connected SSDs for the hottest data blobs. Data routing problems are only one face of the medal: all this data needs to be elaborated and custom software has to be designed with heavy parallelization in mind. On the computing side, elaboration hardware is to be specifically tailored for the application needs. These aspects needs to be evaluated by professional figures with prior expertise in the field to not result in catastrophic bottlenecking.

Chapter 5

A Biological Framework

The brain is a world consisting of a number of unexplored continents and great stretches of unknown territory.

Santiago Ramon y Cajal

The work of 3D neuronal segmentation is set in a well-developed brain research context that branches off in a large variety of approaches: a pre-condition to every single one of those methodological views is a basic - at least - comprehension of the underlying biological framework. In this section we're going to remark some basic neurobiological aspects to get a firmer grasp of the problem, explore a few problematics and nodal points in brain tissue imaging and then we'll proceed to a quick overview of some scientific approaches - other than fluorescence microscopy - that serve us as a baseline to compare results.

5.1 Neural Tissue

The vertebrate nervous system is formed by the Brain, the Spinal Cord, which together compose the so-called Central Nervous System, the Sensorial Organs, and by all the nerves that interlink those organs with the rest of the body, composing the Peripheral Nervous System. Human brain tissue is composed by an order of 10^{10} cells which can be distinguished in two main classes: **neural cells**(**neurons**) are highly differentiated cells which can transmit electrical and chemical signals while there are many types of cells that serve as structural and functional support for

the neurons that go under the name of **glial cells** or **neuroglia**[1]. While neurons are characterized by membrane excitability and conductivity, making them fit to conduct signals using a membrane depolarization wave mechanism, glial cells don't share such properties.

5.1.1 Neurons

Each neuron is composed of a cell body, the **soma**, that hosts the nucleus many cytoplasmic organelles, short cytoplasmic processes known as **dendrites** which passively conduct neuronal impulses coming from other cells, and a long cylindrical projection that actively propagates signals from the cell body to other neurons, the **axon**.Neurons can communicate via some specialized membrane regions called **synapses** that are located at the terminations of both the axon and the dendrites.

In the central nervous system neural tissue shows a well-defined histological configuration: the neuron somas, along with their dendrites and the initial tract of the axon that's not embedded in myelin (see below in the glyal cell section), aggregate to form **grey matter** which is localized in the outermost part of cerebral and cerebellar hemispheres (called **cortex**), in the central part of the encephalon and in the spinal cord. Axons embedded in myelin that emerge from the gray matter form the *white matter*, which occupies the central part of cerebral and cerebellar hemispheres and the external part of the spinal cord.

Neurons present themselves in different forms, varying in number and ramifications of the cytoplasmic processes, length of the axon and soma's dimensions (ranging from $4\mu m$ in cerebellar granule cells to $100\mu m$ for motor neurons in the spinal cord) and can be classified by different criteria.

By classifying them relatively to cellular body morphology and ramification of the cytoplasmic processes we can find four different categories, as depicted in figure 5.1:

[1] the name originates from the Greek $\gamma\lambda\iota\alpha$ "glue", which suggest initial assumptions of a structural-only function of those cells, it is later discovered that architectural purpose is just is just one of the many roles that this class of cells play.

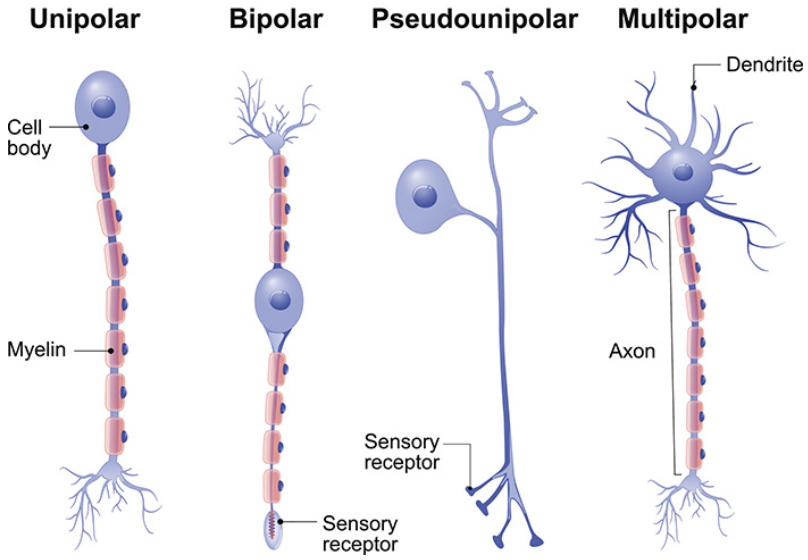

Figure 5.1: Neuron Shapes [120]

1. **Unipolar neurons** They have a single extension, mainly present in the mature human body as *photoreceptive neurons* and *olfactory neurons*.

2. **Bipolar Neurons** They have two extensions: an axon and a main dendrite, in humans are present in *cochlear ganglion* the *vestibular ganglion* an in the *retina*.

3. **Pseudo-unipolar Neurons** Spherically shaped, they have a main extension that generates from the cellular body and then ramificates in a T shape, a branch directed peripherically and a branch going towards the CNS, both processes share the same morphological axon properties but have different functional behavior: one acts as a dendrite, conducting neural impulses *towards* the cell body, the other acts as an axon, conducting *from* the neural body to the CNS. Such neurons can be find in *cranial nerve ganglia* and in the *roots of the spinal nerves*.

4. **Multipolar Neurons** They have many dendrites, allowing connection to many other neurons and a single axon, they are the most numerous in the CNS and can have different shapes: pyramidal shape, as the *pyramidal neurons* in the cerebral cortex, flasklike as *Purkinje neurons* in cerebellar cortex

and star-like as the *motor neurons* in the spinal cord.

5.1.2 Neuroglia

Neuroglial cells are 10 times as many as neural cells and form a dense network that embeds bodies and processes of the neurons, providing both mechanical and a metabolic support. They are classified as *astrocytes*, *oligodendrocytes*, *microglia* and *ependymal cells* in the CNS, *Schwann cells* and *satellite cells* in the PNS. Since our areas of interest are located in the CNS we're going to spend a few words on the glial cells found there, overlooking PNS glial cells.

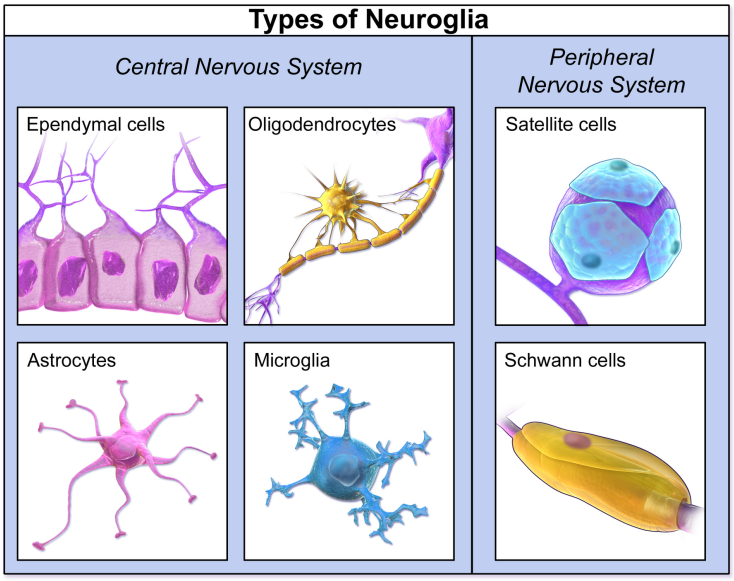

Figure 5.2: Different types of glial cells

Astrocytes

Astrocytes form the largest cell population amongst glial cells, they present $8\mu m$ wide cell bodies and many cytoplasmic projections that link neurons to their blood supply while forming the *blood-brain barrier*. Astrocytes influence the chemical environment of neurons by removing excess potassium ions and take part in the

metabolism of *glutamate* and *gamma-aminobutyric acid* (or $GABA$) neurotransmitter, removing them from synapses and methabolizing into their precursor amminoacid: glutamine. It has been shown that astrocytes, although they can't produce long-distance signals, can "communicate" by opening membrane channels for Ca^{2+}: neuronal activity can directly influence astrocytes and they can influence back neurons by regulating local blood flow.

Oligodendrocytes

Oligodendrocytes show slightly lower dimensions than astrocytes and are characterized by few cytoplasmic processes. There are two different classes of oligodendrocytes: *satellite oligodendrocytes* are present in grey matter, in close contact with neuron somas and processes while *interfascicular oligodendrocytes* are found in white matter, forming row structures between axons and myelin fibers. The main function of oligodendrocytes is to to deposit myelin around axons in the CNS, providing insulation of the axon itself and allowing for better signal propagation.

Microglia

They're the smallest glial cells and actually are specialized macrophages characterized by small and irregular cellular bodies with many thin and short dendritic extensions. They're observed to increase in number in correspondence of damaged tissue areas: they're capable of cytokine and hydrolytic enzyme production and their function is to protect neurons of the CNS by phagocytosis of damaged or degenerate neuron fragments.

Ependymal cells

Ependymal cells - or *ependymocytes* - form a lining of ventricular cavities of the encephalon and the spinal cord central canal. They're involved in the creation and secretion of *cerebrospinal fluid*: within the brain ventricles a population of ependymal cells and capillaries form a structure, called the *choroid plexus* which is responsible for the production of the *cerebrospinal fluid*.

5.2 Anatomical framing

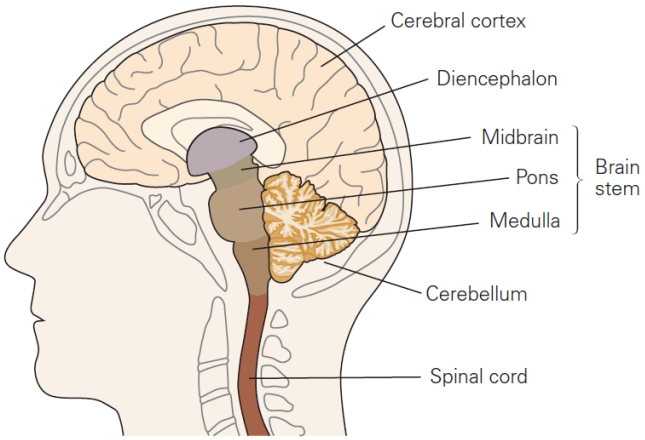

Figure 5.3: Main regions of the Central Nervous System [57]

The central nervous system is composed by the the brain and the spinal cord. The spinal cord can be considered the simplest part of the CNS: it extends from the skull base roughly to the first lumbar vertebra, it receives sensory information from the skin, the muscles of the torso and the limbs and contains motor neurons that arbitrate voluntary movements and reflexes. Our subject of interest, though, is located in the brain. Brain and can be divided into six distinct regions: the *medulla oblungata*, the *pons*, the *cerebellum*, the *mesencephalon*, the *diencephalon* and the *cerebral hemispheres* or *telencephalon*, we're particularly interested in the last. The cerebral emispheres are the widest area of the CNS, they comprehend the *cerebral cortex*, the white matter underneath it and three deeper formations: the *basal ganglia*, the *amigdala* and the *hippocampus*. Cerebral emispheres are involved in sensory, cognitive and motor functions as well as in memory and emotion, they're linked only by the *corpus callosum*. The cerebral cortex, that is the outmost region of the cerebral emispheres, is responsible for higher functions such as the planning of actions, it's divided into four lobes: a *frontal lobe*, a *parietal lobe*, a *temporal lobe* and an *occipital lobe*.

The Neocortex

The outermost region of the cortex and the closest to its surface is called the *neocortex*: here happens most of the neural activity and from here comes the tissue we're analyzing with CNN segmentation techniques. The neocortex is organized in vertical columns and each of them is organized in six different layers, progressively numbered from the outermost layer to the innermost, in direction of the white matter. We can see a schematic representation in figure 5.4.

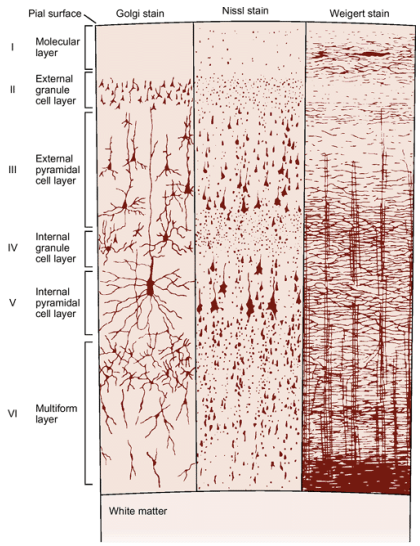

Figure 5.4: Layers of the Neocortex with different staining methods [57].

- **Layer I** Called the *molecular layer*, has very few neurons and cells in itself and mainly contains dendrites from lower layer cells and axons making connections to other cortex areas.
- **Layer II** Mainly contains granule cells and it's then called *external granule cell layer*, its function is to receive inputs from other areas of the neocortex.
- **Layer III** Called *external pyramidal cell layer*, it's made up of pyramidal cells: neurons in the lower part of layer III are generally bigger than those in the higher part of it. Along with neurons of layer II, these neurons mediate intracortical communication as their axons generally project to other neurons of the same cortex areas.

- **Layer IV** Contains a large number of small spherical neurons so it's named *internal granule cell layer*. Granule cells in this layer receive sensory input and relay it to adjacent neocortex column. In the primary visual cortex it's thick enough to be divided into three subsection.
- **Layer V** It's called *Internal pyramidal cell layer* and it's populated by pyramidal cells: these pyramidal cells are bigger than those in layer III, their axons create the principal outward links of the neocortex and projects to other cortical areas and subcortical structures.
- **Layer VI** Many different neurons form this layer which is then called *Multiform layer*, it's the closest to the white substance and it's responsible for receiving and integrating information from the brainstem.

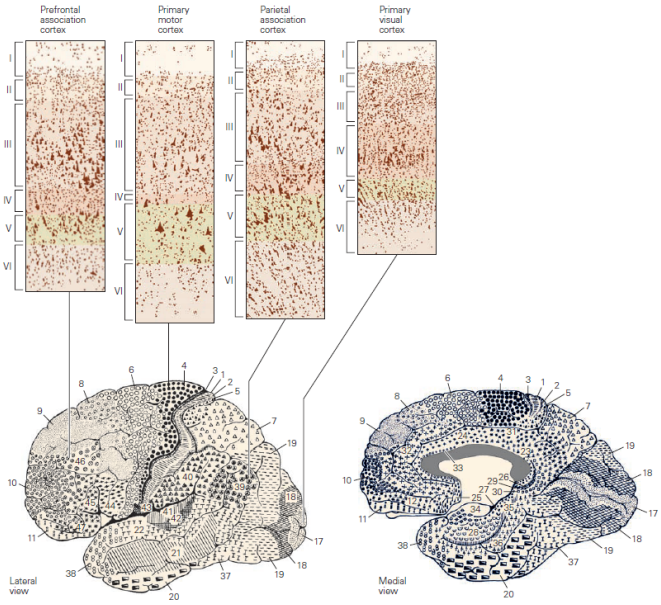

Figure 5.5: Brodmann map and neocortex layers in different cortical areas [57].

The thickness of the layers and the detail of their functional organization varies thoughout the cortex: using as criteria cell dimensions, packing and relative prominence of layers above and below layer IV, in 1909 Korbinian Brodmann - early student of the cerebral cortex - divided the cerebral cortex in 47 different regions

[13]. These divisions, known as *Brodmann maps* have been corrected, refined and improved in the years and are the most widely known and cited cytoarchitectural organization of the human cortex. In figure 5.5 we can see a Brodmann map along with sections of the neocortex in different cortical areas.

Inside the neocortex information exchange happens via both *feed-forward* and *feedback* connections, depending on the specific cortical area. Neocortical neurons are often organized in column structures traversing layers. Every cortical column has a diameter of millimeter fractions and neurons from the same column tend to share similar reaction properties, suggesting that they are part of local information elaboration networks: columns are thought to be the fundamental computational modules of the neocortex.

5.3 Studying the Tissue

There are many ways of observing the tissue involving completely different technical approaches and each one of them helps to shed light upon particular aspects. A common approach is embed the tissue in a solid material that support thin sectioning, slice the tissue itself using a microtome and successively stain the obtained sections for optical study. Microtomes can be conceptually as simple as a razor blade Most microtomes share the same basic functionality having a cutting edge with different properties depending on the material it's made of, a specimen holder and a system of relative positioning of the specimen to the knife. These methods come with the disadvantage of deformation of the tissue, other than the inevitable destruction of the sample.

5.3.1 Staining

Since most cells are essentially transparent, contrast is generally introduced using staining techniques.

"Classic" Techniques

Routine histology uses a combination of *hematoxylin* and *eosin*, commonly referred as **H**&**E**. Hematoxylin is a basic stain with a purple color and stains basophilic

structures like chromatin and the ribosomes, while eosin is an acidic stain with a red color that stains acidophilic parts of the tissue. There is a high number of techniques other than H&E, each one makes it possible to highlight different structures and properties of the tissue. Looking back at figure 5.4 we can see some examples of "classic" staining techniques: Golgi staining is a silver-based staining method that uses potassium dichromate and silver nitrate to induce microcrystallization of silver chromate inside the neurons to make axons and dendrites optically visible, Nissl staining uses basic dyes to highlight negatively charged nucleic acids to mark cell bodies.

Immunohistochemistry (IHC)

Immunohistochemistry staining techniques exploit antigen-antibody binding properties to selectively identify specific proteins in the tissue. Antibodies can be conjugated to enzymes that catalyse a color-producing reaction: this is the case of *immunoperoxidase* staining in which an antibody is tagged to the enzyme *peroxidase*. An antibody can be tagged to a fluorophore instead, such as fluorescein: this makes the tagged proteins visible in fluorescence microscopy.
Tissues studied with the two-photon apparatus were treated with a *NeuN* antibody,

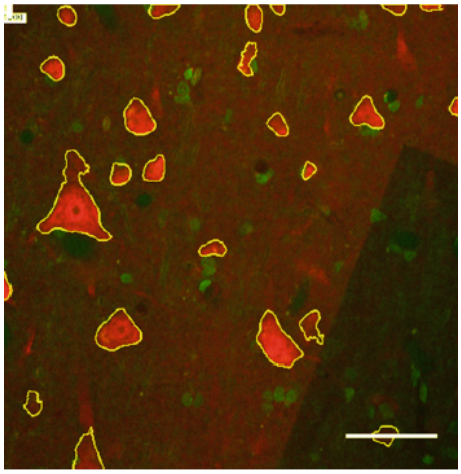

Figure 5.6: TPFM segmented view of brain tissue. [80]

that recognises the NeuN protein, present in neurons, and identify their somas: this marking is the one seen by the CNN model. Neural tissue was also treated with a

secondary marker, ab150080: the combined use of different markers offers a different perspective on the tissue and helps in the human labeling phase. Multistaining can be integrated in a multi-view model as part of a future development stage. In figure 5.6 we can see an example of an already segmented (yellow contours) view of a tissue sample in two-photon microscopy from [80] paper: the *NeuN* marking is represented by the red channel of the image while the green channel traces the secondary marking.

5.3.2 Tissue Clearing

Mechanical sectioning of the tissue, other than inevitably leading to the destruction of the sample itself and introducing significant structural deformations, is a time-consuming process: serial acquisitions for tridimensional reconstruction of a tissue volume is, in most cases, straight up not feasible.
Optical sectioning of tissue is normally hampered by opaqueness of the tissue itself: some form of clearing is to be implemented to render the sample optically accessible. Tissue opaqueness is due to both absorption and scattering events, though the latter is to be considered the principal component. Biological samples are largely constituted by molecules characterized by high refractive index, such as lipids and proteins, while they're immersed in relatively low refractive media like water.
Refractive index of *dry* tissue can be estimated to be around $n_{dry} = 1.50$ which is considerably distant from the refractive index of water $n_{water} = 1.33$: the solution strategy is to reduce as much as possible the mismatch between refractive indexes either by removing some components, like lipids, or modifying their optical properties. Final refractive indexes n_{clear}, according to the particular technique used (more than 20 approaches have been proposed in literature during the years), can range from 1.38 to 1.56. The particular clearing method choice is not a trivial one as one has to carefully weight pros and cons of every procedure, aiming to both mantaining high transparency of the sample and preserving endogenous fluorescence or, alternatively, allowing staining methods like IHC. Also, these evaluations must take into account possible photobleaching or photoquenching effects.
Different categories of approaches are possible, one of them is the immersion of the sample in a mixture of benzyl alcohol and benzyl benzoate, simply known as *BABB* (Benzyl Alcohol Benzyl Benzoate), since BABB is unmixable with water,

previous accurate dehydration of the sample using ethanol is required. Since organic solvents are in some cases observed to cause fluorescence quenching, protein-friendly acqueous environments are sometimes preferrable and many water-based solutions with moderate ($n < 1.48$) refractive indexes are available.
It's possible to simultaneously tackle the problem from the other side by lowering the effective refractive index of the proteins or by removing lipidic components using polyalcohols. Lipid removal can result in protein content loss: *tissue transformation techniques* have been developed to mantain proteins by crosslinking them to some form of gel mesh. The proven state of the art in this regard is represented by the *CLARITY* method [21] and its derived procedures: protein structures are placed into a polyacrylamide hydrogel scaffolding so they remain in place when lipids are removed by detergents, the sample is then submerged in a refractive index matching solution to render it transparent. A more recently derived method named *SWITCH* replaces the polyacrylamide mesh with a more stable glutaraldehyde one that allows for more efficient IHC staining. In the original procedure a commercially available

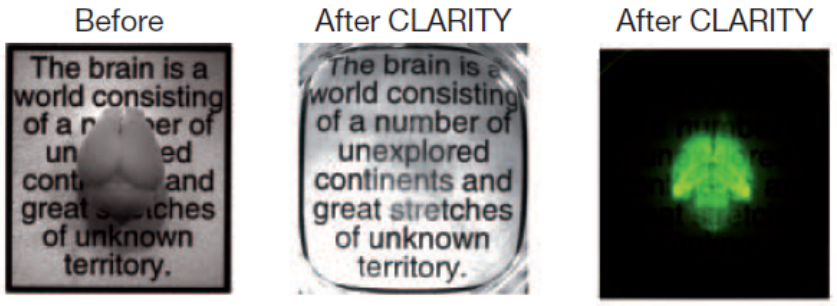

Figure 5.7: Mouse brain cleared with CLARITY procedure and matched with a FocusClear solution [21]

solution with $n = 1.45$ named *FocusClear* - the composition of which is a trade secret - is used but cheaper "open-source" alternatives are available: an example is *2,2'Thiodiethanol* - or *TDE* - with $n = 1.42$ which has been proposed as an alternative solution [24].
There's no "absolute best" clearing approach but the most adequate methodology is dictated by a number of factors, including the type of involved sample, the kind of target to observe (are we observing endogenous fluorescence, like in the case of

transgenic mice, or are we immunostaining an ex-vivo sample?) and the instrumental or economical limitations.
LightSheet Fluorence Microscopy typically uses 1PE illumination, linearity of excitation makes the whole technique sensitive to light scattering inside the sample so the clearing technique choice should take into account requirements for whole sample uniform high transparency. Confocal two photon microscopy on the other hand is much more sensible to optical aberrations, requiring specific lenses corrected for the refractive index of the chosen solution and it's greatly limited by the imaging speed of point-by-point image reconstruction. Long-term stability of the sample is to be taken into account, but at the same time nonlinear excitation improves localization and relaxes requirements on high transparency.

5.4 Examples of Other Imaging Approaches

Fluorescence Microscopy is not the only imaging approach, comparison between fluorescence imaging and other imaging techniques is useful to get a baseline to measure results against: here we introduce two methods that can possibly serve as large-scale references.

5.4.1 Magnetic Resonance Imaging

Important results in brain imaging and tractography have been achieved by Magnetic Resonance Imaging **MRI** and Diffusion Magnetic Resonance Imaging **DMRI**, these methods are extremely useful to large-scale brain analysis of living patients with virtually no detrimental consequences and are routinely used for diagnostic purposes. On a finer scale, huge quantities of human brain fiber tracts data have been collected during the last 15 years giving a substantial comprehension of large-scale connections in the human brain, this data can possibly be integrated with the microscopic data that Fluorescence Microscopy is creating a multi-scale comprehension perspective of the brain inner workings.

5.4.2 Optical Coherence Tomography

Optical Coherence Tomography (OCT) can provide volumetric reconstruction of tissue with micrometric resolution and video-like rate speeds. Optical imaging such as fluorescence microscopy can be complementary coupled with *Optical Coherence Tomography*: OCT has been utilized for optical brain tractography resolving up to $1\mu m$ resolutions exploiting intrinsic optical contrast in all three dimensions and, combined with optical clearing, can be used to resolve myelinated fibers both *in vivo* and *ex vivo*.

Chapter 6

CNN Models and Methods

Is learning better networks as easy as stacking more layers[1] *?*

He et al. - Deep Residual Learning for Image Recognition [44]

Two different classes of Convolutional Neural Network models were explored: a 2D Fully Convolutional model in a 2.5D segmentation setup and a native 3D CNN model. The two networks share the same architectural intuitions as they're inspired, respectively, by Ronnenberger's original U-NET [99] and 3DU-NET [133] by Cicek et al., which represent the current state of the art in CNN-based image segmentation.

An improvement to 3D UNET is then proposed by integrating the problem in a *Residual Learning* framework [44].

Both 2D and 3D CNN models work in a *patch-based* mode, meaning that they're not exposed to the entire image field but only to relatively small patches (2D squares in the first case, 3D cubes in the other case). This approach has been chosen for two order of reasons: one related to resource availability and the other to model versatility.

Trivially, the first reason is that training a convolutional model with large inputs is too computationally expensive for consumer hardware like the machine I used in this work as all the model's tensors (both the inputs, outputs and training variables) are to be allocated in single-digit gigabytes VRAM GPU memory.

On the other side adopting a patch-based approach allows us to decouple model

[1] Spoiler alert: it's not.

input sizes from the actual input image size and, if a good patch merging strategies are implemented, the model can be scaled to arbitrarily large inputs in a size-agnostic way. This setup effectively produces an intermediate approach between the *sliding windows* philosophy described in section 2.1.1 and the "pure" *fully convolutional* one.

Another focus in this chapter will be the a 3D patch-based reconstruction algorithm I coded specifically for this context that aims not only to fuse together patches while suppressing border artifacts, but also to reduce prediction noise by exposing the network to multiple transformed views of the same input volume and averaging the responses.

6.1 2D Model

The 2D network model serves as a baseline for 3D CNN reconstruction and it's based on U-NET, which we introduced in 2.1.6, with a few structural changes.

6.1.1 Architecture

The model is designed to work with 64×64 wide single-channel images, and is composed of a total of 52 different layers, organized in three sections:

- a *contraction* phase made of 4 consecutive downsampling blocks
- a single-block *bottleneck* phase
- an *expansion* phase made of 4 upsampling blocks

The **downsampling** blocks are composed as

- a *Convolution* layer, with a n_{filters} convolutional filters, a $3x3$ kernel and a $ReLU$ activation function
- a *BatchNormalization* layer
- another *Convolution* layer with same characteristics as the previous one
- a 2×2 *MaxPooling* layer

where the number of convolutional filters n_{filters} increases with the layer depth: 16 filters for the first stage and, respectively, $32, 64, 128$ filters for the subsequent blocks.

The **bottleneck** phase is made of

- 256 *Convolution* filters with a 3×3 kernel with *ReLU* activation function
- a *BatchNormalization* layer
- 256 other *Convolution* filters, same characteristics as the first layer
- another *Batchnormalization* layer

The network decontracts features through **upsampling** blocks that are made of:

- a 2×2 *Transposed Convolution* layer, with same number of filters n_{filters} as the symmetrical downsampling block in the contracting phase, that upsamples the features
- ta *Concatenation* with the BatchNormalization output in the symmetrical block
- a *BatchNormalization* layer
- a *Convolution* layer with n_{filters} 3×3 filters and *ReLU* activations
- a *Dropout* layer with same k_{drop} dropout ratio as the symmetrical downsampling block
- another *Convolution* layer, same specifics as the previous one
- a *BatchNormalization* layer

The model's output, lastly, goes through a *sigmoid* activation function so that the final output is a float image with values in [0,1].

Convolution Paddings: VALID vs SAME

There are some structural differences between this model and the one proposed in [99]: in the original work convolutional filters were applied in an unpadded mode meaning that at every convolutional step border pixels of the feature maps were lost, this, on one side, made the network's outputs smaller than the inputs, and on the other made it necessary to use cropping strategies during the concatenation phases.

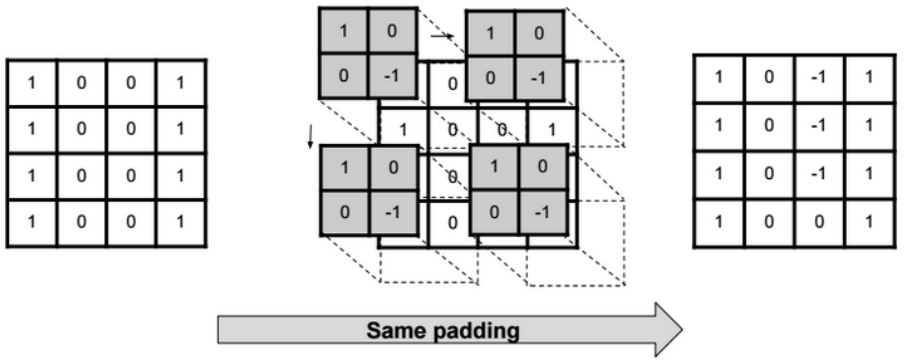

Figure 6.1: SAME Padding

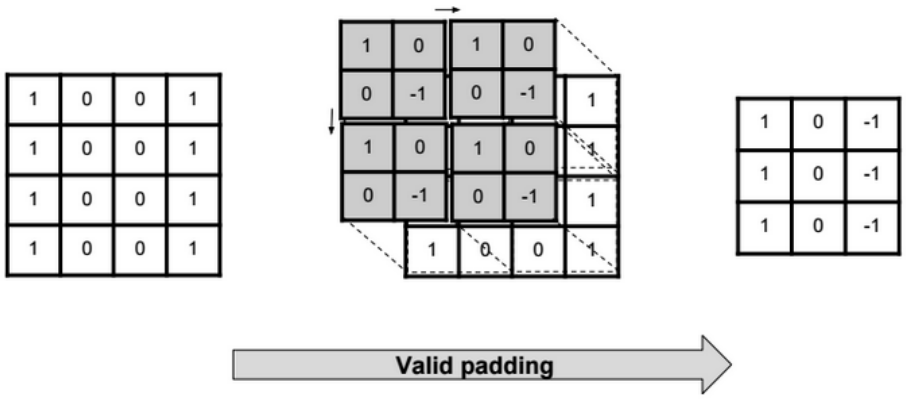

Figure 6.2: VALID Padding

The original paper does not go into much detail on the reasons behind the choice of "VALID" convolution strategies instead of "SAME" ones but they can be inferred easily: in a "VALID" convolutional operation only *real* pixels are part of the computation, meaning that there are potentially no artifacts due to "spurious" values

being considered in the computation. Zero-paddings, on the other side, by artificially introducing zeros at the image borders, tend to produce spurious results in those pixels of the final feature map that are exposed to less information, resulting in border effects. While the use of unpadded convolutions and cropping possibly reduces border effects in the outputs, the patch-based implementation I envisioned for this setup made me prefer to address this problem at the reconstruction stage rather than embed it structurally. Using "SAME" paddings made it possible to achieve a better architectural symmetry, letting symmetric blobs directly concatenate without introducing any cropping strategy. Another advantage of using padded convolutions is that outputs could be made to have the same size of the inputs, making it algorithmically easier to obtain full-scaled predictions over extended inputs. More on this topic will be covered in the Reconstruction Strategies section.

6.1.2 Data Augmentation

The above depicted model, once compiled, leaves us 905.681 trainable parameters: a zero-order statistical evaluation would find it reasonable to assume that model convergence would need an amount of training examples that's similar to the number of free parameters. An exaustive statistical discussion of model convergence in terms of free parameters and effective examples would be both arduous and lengthy, the reader will be redirected on specialistic tractations of such themes for such kinds in-depth analysis: we limit ourselves to point out that, even if Convolutional Neural Networks variances can converge with way less examples than the number of free parameters, thanks to both regularization strategies and intrinsic a-priori assumptions on the data distributions that can be though as hard-coded on a structural level, the number of labeled examples provided in a dataset is rarely large enough for a model to successfully train without any data augmentation. Limited quantities of segmented data require us to virtually extend the dataset in a statistically meaningful way by developing a data augmentation pipeline.

In the 2DCNN case a pipeline for data augmentation was created: at each training step the training examples had indipendent probabilities to undergo one or more of these transformations:

Horizontal Flip : inversion of the horizontal coordinates of the image.

Vertical Flip : inversion of vertical coordinates.

Random rigid rotation : random rotation by an integer multiple of 90°

Transposition : exchange of vertical and horizontal coordinates.

Random Brightness and Contrast Adjust : application of a linear LUT of the type $x_t = a * x + b$ where a is randomly sampled in a range between 0.8 and 1.2 and b is sampled in the range -0.2 and 0.2.

Addition of Gaussian Noise : Addition of Gaussian noise sampled from a distribution with a variance that's randomly selected in a range between 0.01×255 and 0.01×255.

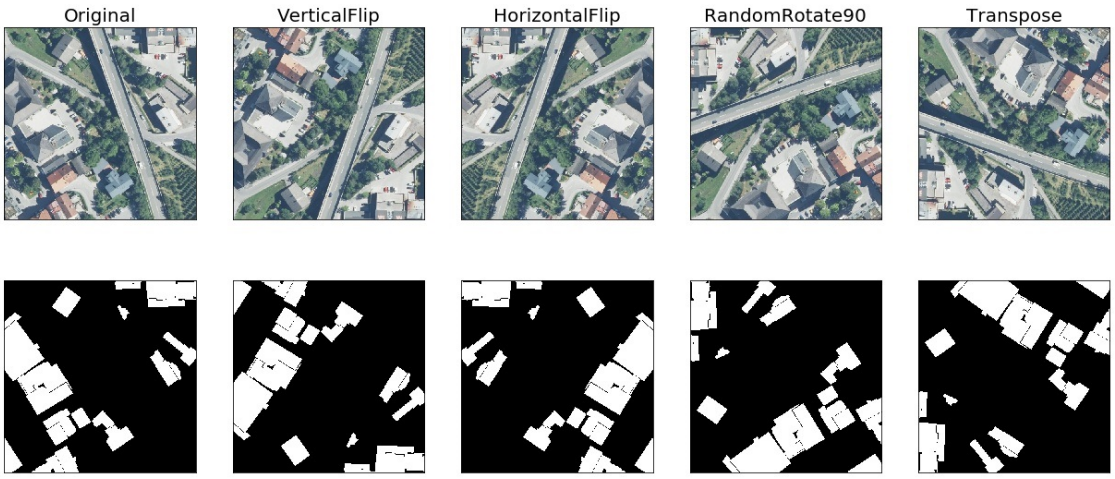

Figure 6.3: Data Augmentation Transformation Examples

More complex transforms are also possible but temporarily discarded in order to mantain acceptable computation times[2] Randomly transforming each image in

[2]Even if computation times in transformations like grid-based distorsions can be added in reasonable times for the 2D images, they can be extremely lenghty for 3D patch augmentations. Since we want both the 2D and 3D model to share most of the premises for a reasonable comparation we try not to give one of the models unfair advantages with respect to the other and try to avoid implementing conceptually different types of data augmentation transforms.

the dataset, at runtime, produces an effective dataset of plausible examples that is way larger than the original. The augmentation process is beneficial to the training as long as the transformed images are likely to have been drawn from the "real" distribution: all the applied transformation should produce results that don't differ from the inputs in critical ways (an example of the validity limitations of such transformations would be the automotive dataset example in 1.6.2).

6.1.3 Training

The model was trained using the *Adam* algorithm described in 1.3.1 using an initial learning rate $\eta = 0.001$ and the terms for the moving averages for gradient moment estimation being set at $\beta_1 = 0.9$ and $\beta_2 = 0.999$. An extra policy was used to sequentially reduce the learning rate η when the validation loss stagnates for more than 5 epochs. The weights were randomly initialized using the methodology descripted in by Glorot and Bengio in [40]: for each hidden unit samples from a uniform distribution within $-l$, $+l$ are drawn, where l is $\sqrt{\frac{6}{f_{\text{in}}+f_{\text{out}}}}$, f_{in} is the number of input units in the weight tensor and f_{out} is the number of output units in the weight tensor for each hidden unit.

The chosen loss function is the *binary cross-entropy* function described in 1.18, while both accuracy and Dice score metrics were monitored during the training.

6.2 3D Model

The 3D Neural Network is modeled after 3D U-NET [133] and tries to exploit the same architectural intuitions of the 2D network in a 3D framework. Resource constraints make it necessary to use reduced the sizes of receptive fields: in this case inputs are monochrome volumes of $64 \times 64 \times 64$ voxels.

6.2.1 Architecture

As in the 2D model, the architecture includes a downsampling a bottleneck and an expanding phase:

the downsampling section is composed of three *downsampling blocks*, each built as a sequential combination of two *Convolutional Blocks* followed by a $2 \times 2 \times 2$

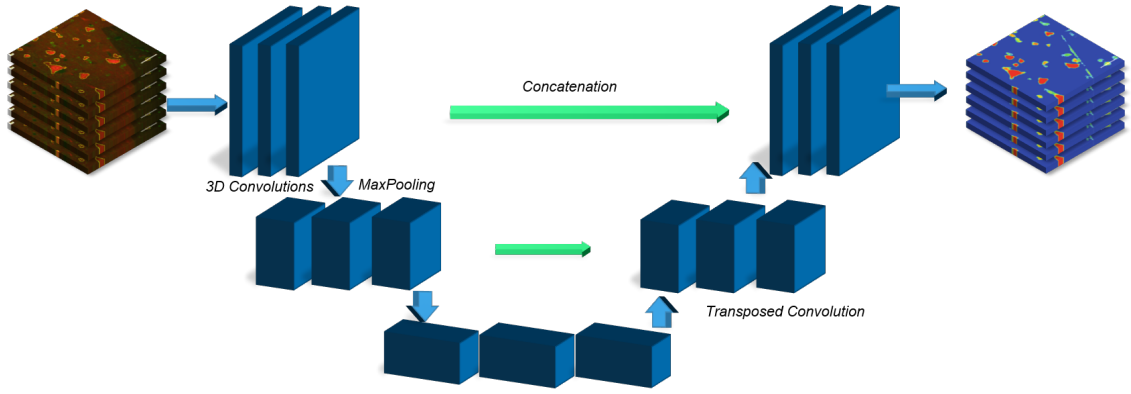

Figure 6.4: 3D CNN Model

Maxpool layer, where I defined *Convolutional Blocks* as a simple sequence of

- A *3D Convolution* layer of the type in 2.2.2 with n_{filters} $3 \times 3 \times 3$ convolutional filters, with "SAME" padding and $1 \times 1 \times 1$ stride
- A *BatchNormalization* layer
- A *Dropout* layer
- A *Relu* activation function

The second *convolutional block* of every *downsampling block* has two times the filters as the first one, the number of filters is derived as

$$n_{\text{filters}} = (2^d) * n_{\text{base}} \tag{6.1}$$

where d is an index representing depth of the block (0 for the first block, 1 for the second, 3 for the third) and n_{base} is a base number of filters, set to 16.

The *bottleneck* part is composed of two subsequent *convolutional blocks* of the above type with $n_{\text{filters}}(3) = 128$ and $2 \times n_{\text{filters}} = 256$ filters.

The *Upsampling* phase is symmetric to the *Downsampling* one, with three *Upsampling* blocks made of

- A *3D Transposed Convolution* layer of the type in 2.1.4 with n_{filters} $2 \times 2 \times 2$ transposed convolution filters, with "SAME" padding and $2 \times 2 \times 2$ stride

- A *Concatenation* with the output of the symmetrical block in the downsampling phase
- A *Convolutional Block* of the type descripted above, with n_{filters} filters
- A second *Convolutional Block*

The last layer of the network is represented by a *sigmoid* activation function that converts the outputs of the network in a [0,1] range.

6.2.2 Data Augmentation

Scarcity of input data is an even heavier problem in the 3D case due to the geometrical scaling of the receptive fields. As in the previous case the input dataset is augmented in a real-time manner using random generators but the processing of 3D cubes instead 2D patches brings additional operation complexity that make complex image transformations like elastic transforms or random grid distorsions too slow for this kind of setting. The resulting pipeline is composed of

Flips : inversion of the x and y axes, the z axis is not inverted because of the vertically oriented nature of neocortical structure

Transpositions : exchange of of x and y axis

Random Rigid Rotations : 3D rotations around the vertical axis

Addition of Gaussian Noise : addition of Gaussian noise drawn from a distribution with variance chosen in a range between 0.006 and 0.01

Random Brightness and Contrast Adjust : in the same fashion as the ones described in 6.1.2.

We acknowledge that implementing heavier forms of data augmentation would be beneficial to the training but limit ourselves to the above methods for computational resource limitations.

6.2.3 Training

There's no substantial difference in the training process between the 3D CNN and 2D CNN models: both use an *Adam* optimization algorithm with a *cross-entropy* loss function. Both models are available for download on GitHub at `filippocastelli/CNN3Dbase`, the chosen framework for both models was `Tensorflow 1.15`, with extensive use of the integrated `Keras` functional API.

6.3 Improving UNETs: Residual Learning

As they are, UNETs and 3D UNETs represent the state of the art in a large variety of medical segmentation tasks, however in the satellite image segmentation field it has been showed that they can be improved by integrating concepts from other successful Deep Learning approaches, namely *Deep Residual Learning* [131]. The term *Deep Residual Learning* appears in 2015 in a paper from He and al. [44] and refers to a simple yet brilliant and innovative approach to training deep networks.

Deep Convolutional Networks' unpaired learning ability is to be ascribed to the sequential abstraction of information that, layer by layer, creates synthetic representations of the input data. One is consequentially prone to logically conclude that, when increasing model depth, representational capacity is to increase as well, possibly boosting performances. Simonyan et al. actually showed a direct correlation between model depth and performance in 2014 [110] that led many to think the whole Deep Learning problem as a *stack more layers* challenge. Unfortunately for the expectation of many, training very deep neural networks almost inevitably led to vanishing gradients and to accuracy degradation problems: the accuracy gets saturated and then rapidly degrades in successive epochs in a way that's not traceable to overfitting, and adding more layers just worsens the problem, leading to higher and higher training errors. The idea that a larger network seems to be more difficult to optimize than a shallow one seems to be a rather counterintuitive one (you would expect that with more representational capacity the model would be prone to rapidly overfit to the data) but the mere fact that the degradation problem exists is actually a suggestion -and a very profound one indeed- that there might be intrinsic and deep-rooted differences between training shallow and deep models.

With a bit of mental abstraction we can consider a shallow model, made of a few layers, that solves a classification problem. An indefinitely deep model that solves the same exact problem can be built by adding an number of *identity mapping* layers in which the outputs are just identical copies of the inputs, so at least one solution for the problem has to exist: the one where all added layers are identities.

While this idea alone is not enough to prove that identity mappings are indeed optimal solutions in the real world (it's quite unlikely that the optimal solution for layer happens to be an exact identity mapping), the ineluctable appearance of the degradation problem might suggest that gradient descent solvers have problems at creating identity mappings by finely tuning multiple nonlinear layers.

The simple but brilliant idea behind the paper is the one of flipping the problem by making the network learn the layer functions in terms of perturbative solutions of the identity mapping, or, in their terms, letting the layer optimize *residual mappings*: the underlying concept being that learning to zero-out weights of the residual mapping should fundamentally be easier than optimizing a number of nonlinear layers to represent an identity function.

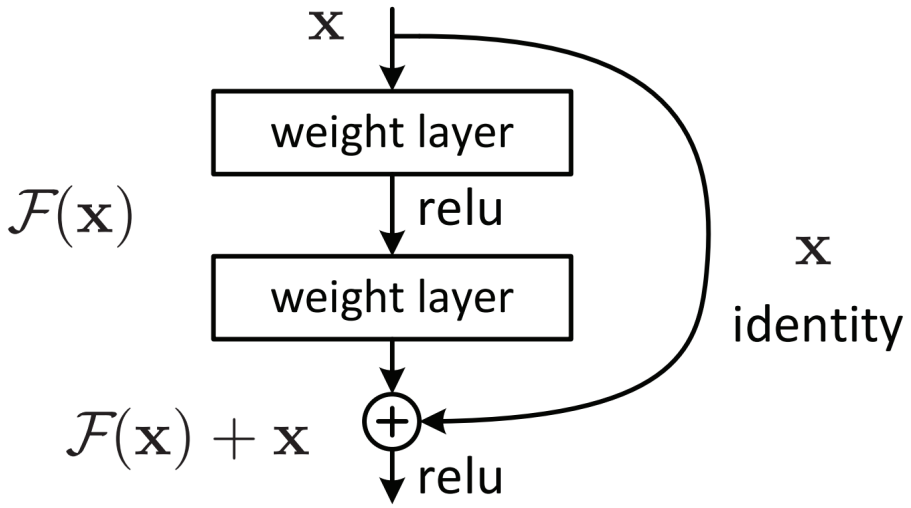

Figure 6.5: Residual Unit [44]

Let's consider a simple layer block with x input for which the optimal solution is $\mathcal{H}(x)$, such as the one in 6.5. We can let the system learn the function $\mathcal{F}(x) :=$

$\mathcal{H}(x) - x$ by directly summing the outputs of the block to its inputs, effectively creating the mapping

$$y = h(x) + \mathcal{F}(x) \tag{6.2}$$

where $h(x)$ represents an identity mapping function. The necessity of defining a mapping function $h(x)$ arises from the consideration that the output of the $\mathcal{F}(x)$ block might not have the same size as x: a simple solution is to transform minimally the input with a linear projection W_s so that $h(x) = W_s x$ and

$$y = Wx + \mathcal{F}(x) \tag{6.3}$$

Note that this can easily and efficiently be implemented with a bias-less 1×1 convolution. The representational capability of the resulting block is slightly increased because of additional the free parameter introduced by the projection: the projection can be avoided if input and output sizes are compatible.

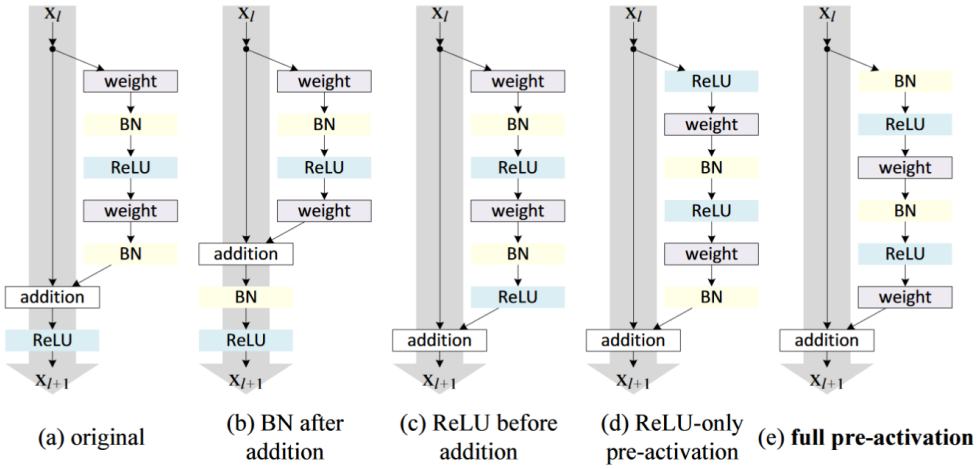

Figure 6.6: Different Positions for Activation Function [45]

What would be the best configuration for a similarly constructed block and how should convolution blocks (or fully connected in the original paper), activations and batchnormalizations be arranged? A common practice for convolutional neural networks is to place batchnormalization layers and activation functions *after* the convolution block, but, when considering a *plain* network with N convolution layers and $N - 1$ activations, it isn't particularly binding if we think activations as

being placed *before* or *after* convolution blocks as long as they regularly alternate: a *post-activation* of a layer can easily be seen as a *pre-activation* by the next one. In residual blocks like the one depicted in 6.5 this approximate isotropy is broken by the elementwise sum and, as a consequence, not all the convolution-activation configurations are equivalent.

The same authors compared different combinations of transformation blocks, batch normalizations and activations [45] and measured that the most effective arrangement is the one depicted in figure 6.6 (e): the classic *post-activation* approach in this new setting proves itself to be ineffective as placing a $ReLU$ activation right before the addition forces the transform $\mathcal{F}(x)$ to output only non-negative values (see definition 1.31), this harms the representational capacity of $\mathcal{F}$ that can now learn only positive functions

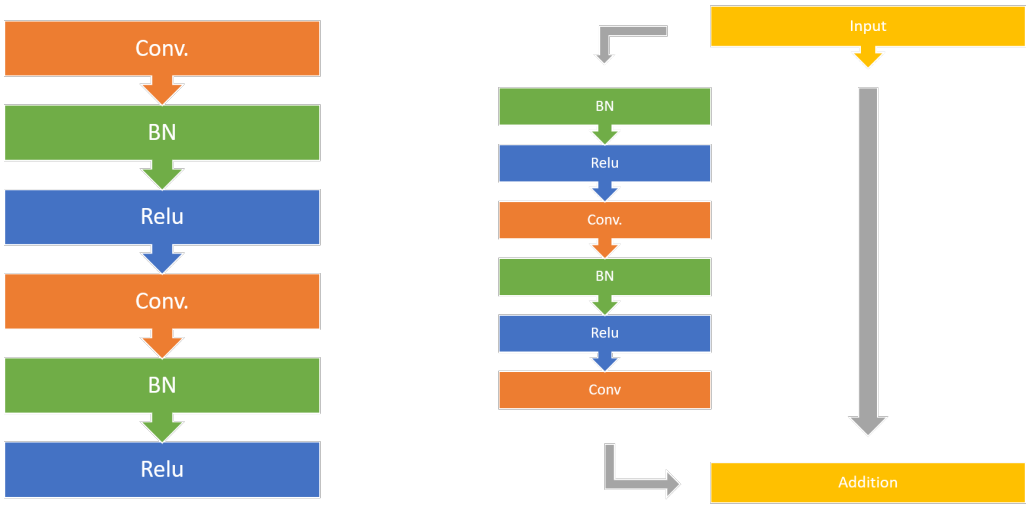

Figure 6.7: UNET Convolutional Block and Residual Convolutional Block

Having introduced the residual learning framework, the only thing that remains is to integrate it into the UNET-like model. The extremely intuitive formulation of residual blocks makes it possible to easily create a substitute for the UNET convolutional blocks as illustrated in figure 6.7: the network's structure remains unchanged, exception made for the basic convolutional blocks, Upconvolution blocks share the same organization of convolutional blocks with the addition of initial

Upsampling and Concatenation steps.

6.4 Patch-Based Reconstruction: SP3D

When it comes to making segmentations of large 3D volumes we must face the fact that the our model accepts only small 64×64×64 patches: to overcome this limitation a patch-based reconstruction strategy of these large volumes needs to be thought. Here we introduce a reconstruction library, coded in `Python`, I developed to solve this problem called `SP3D`, short for *SmoothPredict 3D*. The tool can be thought to be a loosely derivative work of a pre-existing project that applied similar strategies to 2D full-scale image reconstruction, created by GitHub user G. Chevalier [20]. The code is freely available on GitHub at `filippocastelli/smooth_predict3D`.

6.4.1 Border Artifacts

The naive option would be to serially divide the input volume into contiguous $64 \times 64 \times 64$ patches, feeding them one by one (or in batches) through the network and using the resulting prediction volumes to reconstruct a segmented version of the original inputs. This approach usually results in very noisy and discontinuous outputs because of border artifacts of the network: as mentioned in 6.1.1 we use a zero-padding strategy that produces spurious zeros in the inputs at every convolutional level, resulting in observable border effects in the final image. The presence of observable border effects ideally translates to a non-uniformity of prediction accuracy over the output's volume: parts of the prediction that are close to the borders are less accurate than those on the geometrical center.

6.4.2 Patch Blending

One way to address the problem would be making predictions of overlapping patches from the inputs and spatially averaging the results in such a way that every part of the final image would be passed, at least one time, through the "accurate" part of the network's receptive field. This, empirically, produces better results than a non-overlapping approach, but still doesn't completely solve the problem of accuracy non-uniformity over the receptive field. The adopted solution tackles the problem

by introducing a weighted mean over patches by using a 3D window function in such a way that, during the averaging phase, the pixels that are closer to the center of the patch are more relevant than the peripheral ones. The window function is a 3D version of a second-order spline window that smoothly goes to zero to the sides of the input cube and ensures a smooth transition between overlapping 3D patches. In

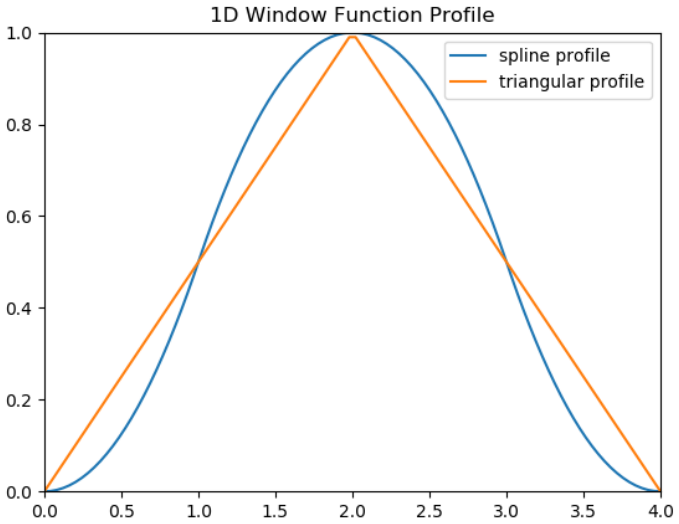

Figure 6.8: Spline Window Profile Compared to Triangular Window

figure 6.8 we represent a 1D profile of the window function compared to a triangular window function profile, such profiles can be shifted and added together, yielding a constant exposure over the whole image, as seen in figure 6.9.

In figure 6.10 we plotted a 2D section of the 3D profile descripted above.

6.4.3 Noise Reduction

Another problem I wanted to tackle in large scale reconstruction was the reduction of noise in the predicted images: to achieve this, additionally to the overlapping patches weighting we introduced an additional strategy. Given that the training data includes a set of geometric transformation invariances (typically the same transformations used in the data augmentation phases), one could assume that transforming the extended input volume in the same way (by rotating or flipping it), the network's

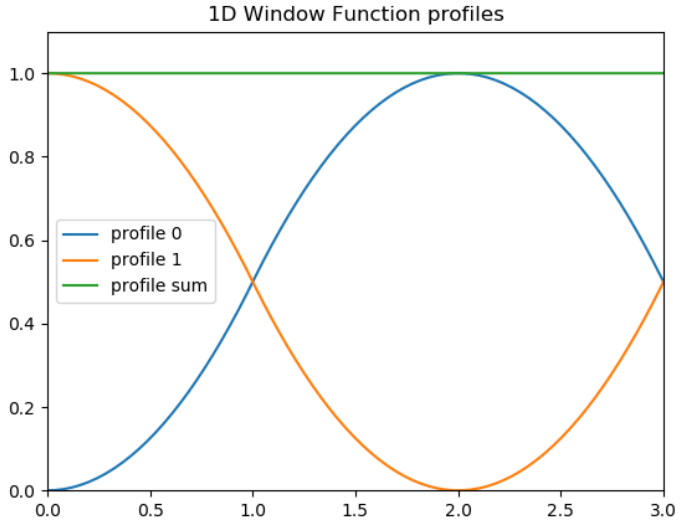

Figure 6.9: Combining Shifted Profiles

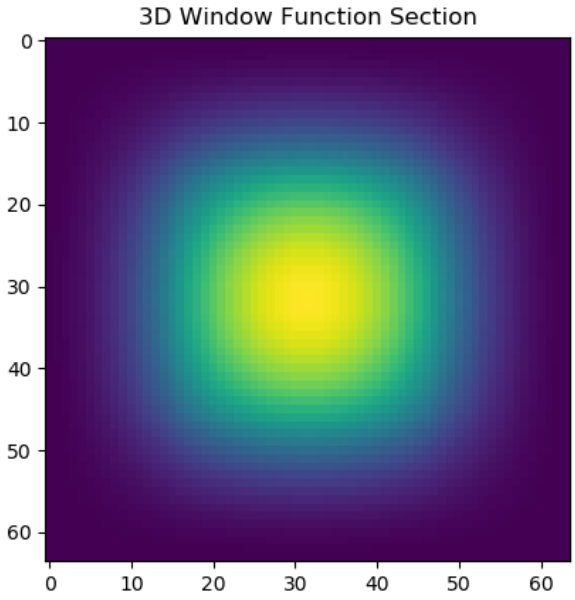

Figure 6.10: Section of a $64 \times 64 \times 64$ Spline Window Function

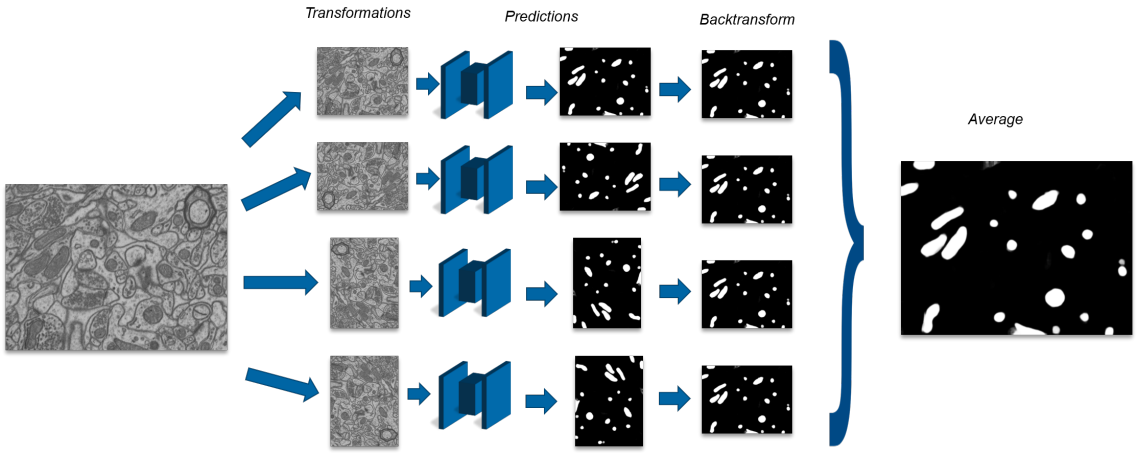

Figure 6.11: SP3D Noise Reduction

output would be the same, just transformed. This reasoning doesn't hold exactly true as there are small differences in the responses depending on the particular orientation of the inputs. We can ideally model the small differences in segmentations due to the inputs under a set of transformations as a noise component such that $\hat{y} = y + \epsilon$ where ϵ is a noise variable that assumes random values in different observations. By averaging over different observations of the predictions (transforming the inputs, feeding the transformed inputs to the patch-based reconstruction system and then back-transforming the output to be in the same format as the input) we should be able to retrieve a better estimate of the "true signal". This is exactly what `SP3D` does: the user defines a set of transformations (flips and rotations) under which the model response should be invariant and `SP3D` creates accordingly transformed versions of the inputs, for each of them it proceeds to create overlapping patches of the inputs, feeds them to the CNN model and reconstructs a single "view" of the large-scale input that it back-transforms, all these single views then get averaged to retrieve a less noisy final output.

6.5 Finding 3D Surfaces: Marching Cubes

At the end of the reconstruction phase we are left with a tensor of values, each representing a class-membership probability at a certain location in space. For most visualization tasks, scrolling sections of the volume as you would do with regular

image stack, might be informative enough but there are particular setups in which it will be more practical to look at the 3D data from different perspectives as a 3D rendered object. The prediction data is in a probability format, with values continuously[3] ranging from 0 to 1. Rendering 3D data requires us to find surfaces that contain values above a certain probability threshold: this is done using a variant of the classical *marching cubes* algorithm by Lorensen et al. [76] that reconstructs probable isosurfaces corresponding to a given threshold level, starting from a point cloud of values.

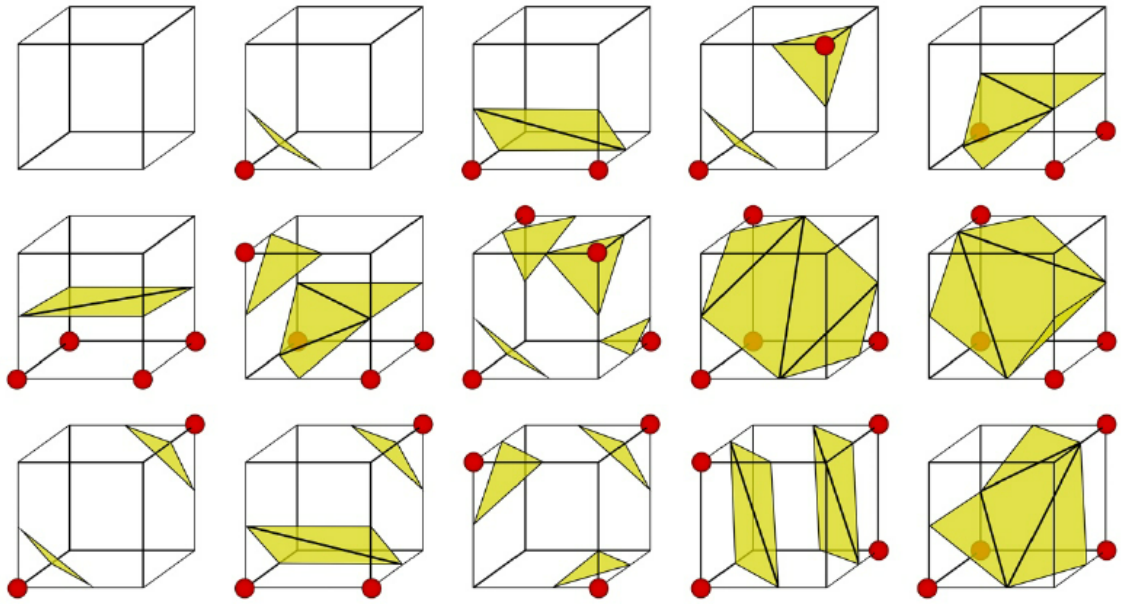

Figure 6.12: Marching Cubes Isosurface Crossing Configurations

Marching cubes is a simple but effective algorithm for estimating isosurfaces of a 3D scalar field as polygonal meshes, it was presented at SIGGRAPH-1987 by Lorensen and Cline [76] and, initially developed for visualization of CT and MRI scan data, in the next years became the go-to algorithm for a very large class of imaging problems. The algorithm's founding principle is extremely simple: by thresholding a the scalar field one would have a lattice of points where the field is sampled, whose binary value would either be 0 (sampled field value under threshold) or 1 (sampled

[3]with float level granularity

value over threhsold). By considering a cube defined by 8 contiguous vertices there are only $2^8 = 256$ inferrable ways in which the cube can be cut by the isosurface with either zero or one intersections on each of the cube's edges. Considering both rotational and mirroring symmetries, the 256 possibilities can be summarized in just 14 distinct cases, shown in figure 6.12. For each vertex the crossing position is estimated using bilinear interpolation with the original scalar values on the vertices. The original algorithm, which actually was under patent until 2005 and could not be used in its original form without paying royalties, has been refined multiple times to better manage ambiguous cases in which multiple surface reconstructions are compatible with the vertex distribution, the particular version we used was proposed by Lewiner in 2003 [70] and comprehends improvements that guarantee topologically correct reconstructions without the possibility of cracks and inconsistencies.

Part III

Results and Conclusions

Chapter 7

Results

7.1 Data Characterization

Two different types of data were used in this work: an already cleaned and optimized Electron Microscopy dataset, mainly for testing and architectural search, and actual Fluorescence Microscopy data acquired at LENS. Working on a "simple" dataset was crucial during the development phases as a way to decouple data cleaning problems from the rest of programming and architectural design challenges.

7.1.1 Electron Microscopy Mithocondria Detection Dataset

The first dataset was created by the Lausanne Polytechnic Computer Vision Laboratory Group as part of their research on automated mithocondria volumetric recognition in the CA1 hippocampus region of the brain [77]. The datasets consists of a $1065 \times 2048 \times 1536$ volume, with voxel resolution $5 \times 5 \times 5nm$, corresponding to approximately a $5 \times 5 \times 5\mu m$ section of hippocampus in the CA1 region. Of this data two $165 \times 758 \times 1024$ sub-volumes are manually annotated: one for training and one for testing.

From a visual standpoint mithocondria are extended objects that present very distinct features and the segmentation task is not an arduous one: the use of a relatively small and "easy" dataset is a significative help when it comes to debugging both the neural network that the reconstruction algorithm.

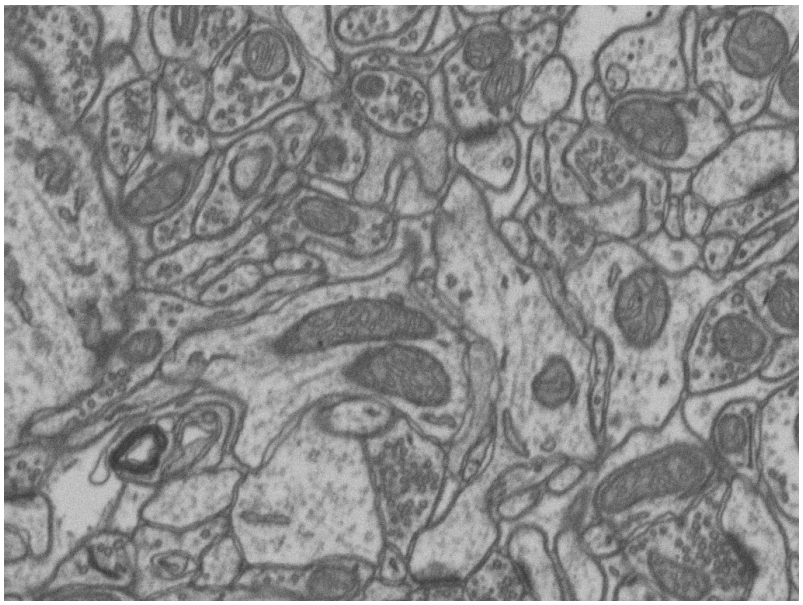

Figure 7.1: Frame from Electron Microscopy Mithocondria Segmentation Dataset

7.1.2 Fluorescence Microscopy Dataset

Fluorescence microscopy images of cortical tissue are a obtained by the Biophotonics group at LENS using both Confocal Two Photon Fluorescence Microscopy and Light Sheet Fluorescence Microscopy techniques. We primarly use stitched data from the TPFM setup: TPFM images come in a multi-channel format, with each channel corresponding to a different marker, of which we use only the first channel, corresponding to the *NeuN* marker. The *NeuN* marker highlights only neuron somas, leaving us an image -like the detail in figure 7.4- that should primarily represent the position of neurons bodies in the tissue. As we can see, these images are significantly more visually complex than the ones in the EM dataset: visual features of the neuron bodies are much less prominent and the acquisition process inevitably produces gradients that are particularly evident in the extended stitched reconstructions.

7.1.3 Data Stitching: `ZetaStitcher`

Both LSFM and TPFM setups produce data in a mosaic-like manner as images are produced in the form of partially overlapping 3D volumes that need to be fused together to obtain large views of brain tissue. LENS developed, specifically for this

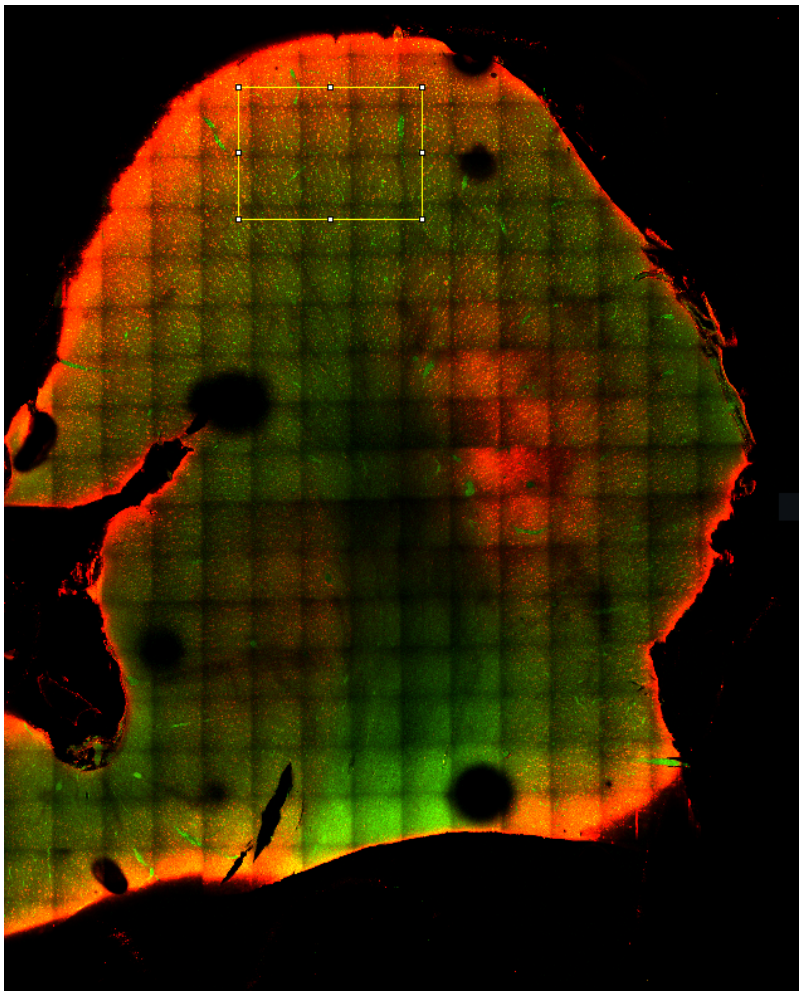

Figure 7.2: Extended Two Photon Microscopy Frame

task, a Python library called `ZetaStitcher` [81] that finds an optimal stack alignment by evaluating cross-correlation of overlapping areas through FFT. The tool creates a symbolic representation of the stitched dataset without actually storing a full-aligned copy of it in memory, a specifically designed API can be used to perform queries to the so-obtained dataset. In this work the models are not directly interfaced with this API and use already stitched copies of the data. In the near future plans are to directly interface stitching, prediction and reconstruction algorithms into a single pipeline.

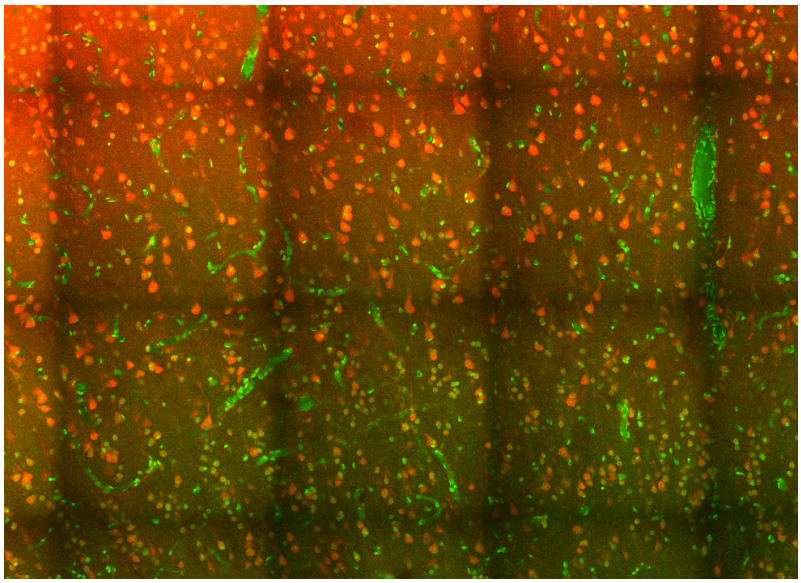

Figure 7.3: Detail of TPFM Frame: Multichannel

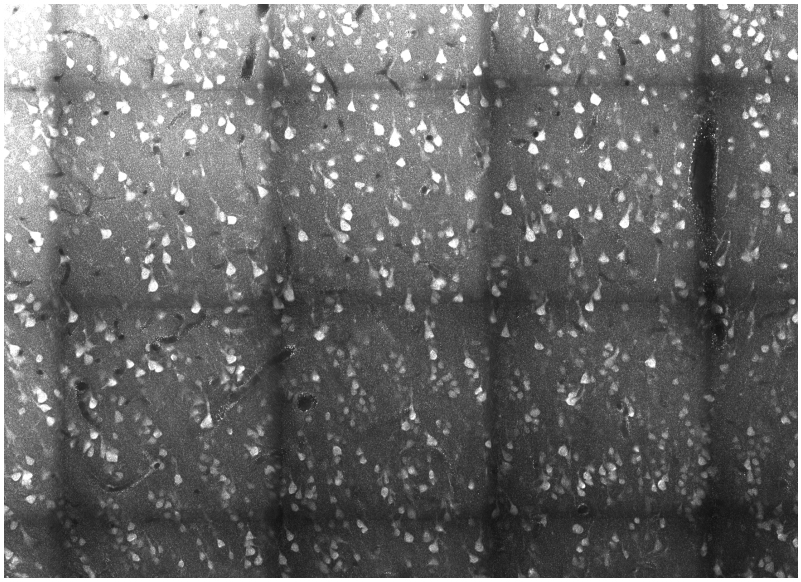

Figure 7.4: Detail of TPFM Frame: NeuN Only

7.2 Evaluation Metrics

In order for us to evaluate results we need to establish some evaluation metrics set first. In the vast majority of cases automatic segmentation is evaluated in terms of overlap measures, such as the ones introduced in section 1.3.3. We will consider metrics that follow the same set-similarity measure idea for evaluating our results. We take as reference *Taha, Hanbury - Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool* [117] from which we derive the following definitions.

7.2.1 Terminology

The rest of the section will be built upon a terminological common ground with a few basic terms. Let the imaging volume be defined on a 3D grid of $w \times h \times d$ size, each point on the grid is called a *voxel* and the whole image can be represented as a set of voxels

$$X = \{x_1, x_2, \ldots, x_n\} \qquad |X| = w \times h \times d \tag{7.1}$$

The *ground truth segmentation* can be defined as a partition $S_g = \{S_g^1, S_g^2\}$ of X with an assignment function

$$f_g^i(x) \begin{cases} 0 & \text{if } x \in S_g^i \\ 1 & \text{if } x \notin S_g^i \end{cases} \tag{7.2}$$

and let the *model segmentation* be another partition of X, $S_t = \{S_t^1, S_t^2\}$, such that its assignment function $f_t^i(x)$ is a function in $[0, 1]$, one could view $f_t^i(x)$ as the probability that x in a member of class S_t^i. In the particular case that $f_t^i(x)$ only assumes discrete values in $\{0, 1\}$ (this can simply be obtained by thresholding), we say that the segmentation is *crisp*, as opposed to the general *fuzzy* case. The classes are conventionally assigned so that $\left(S_g^1, S_t^1\right)$ is the class of interest and $\left(S_g^2, S_t^2\right)$ is the background class. We assume that the membership of a point x sums to 1 over classes

$$f_t^1(x) + f_t^2(x) = 1 \qquad \forall x \in X \tag{7.3}$$

in accordance to the probabilistic interpretation of the assignment functions.

7.2.2 Confusion Matrix

All the presented metrics will be based on four measures, complexively known as the *confusion matrix*: namely the *true positives* (TP), *false positives* (FP), *true negatives* (TN) and *false negatives* (FN).

$$\text{Confusion Matrix} = \begin{bmatrix} m_{1,1} = \text{TP} & m_{1,2} = \text{FP} \\ m_{2,1} = \text{FN} & m_{2,2} = \text{TN} \end{bmatrix} \tag{7.4}$$

These measures are easily defined in the case that we have a *crisp* segmentation (let's say we adopted a threhsold classification rule in which we consider as positively classified every voxel with class membership probability above 0.5, we just transformed the *fuzzy* output of the model into a *crisp* segmentation.) as

$$m_{i,j} = \sum_{r=1}^{|X|} f_g^i(x_r) f_t^i(x_r) \tag{7.5}$$

on a more intuitive basis one could equivalently define the confusion matrix elements as

True Positives : Count of the voxels that are positively classified both in GT and model segmentation.

False Positives : Count of the voxels that are positively classified in segmentation but are negatives in GT.

True Negatives : Count of the voxels that are negatively classified both in GT and model segmentation.

False Negatives : Count of the voxels that are negatively classified in segmentation but are positives in GT.

In the case of *fuzzy segmentations* equation 7.5 can be generalized using a triangular norm

$$\begin{aligned} g &: [0,1] \times [0,1] \to [0,1] \\ g(p_1, p_2) &= \min(p_1, p_2) \end{aligned} \tag{7.6}$$

for modeling the agreement between the two segmentations on the positie classification of particular voxel, implying that the agreement on the same voxel being

background is $g(1-p_1, 1-p_2)$. The disagreement between the two segmentations is modeled by the signed difference $p_1 - p_2$.

The elements of the generalized confusion matrix can then be expressed as

$$\begin{aligned} TP &= \sum_{r=1}^{|X|} \min\left(f_t^1(x_r), f_g^1(x_r)\right) \\ FP &= \sum_{r=1}^{|X|} \max\left(f_t^1(x_r) - f_g^1(x_r), 0\right) \\ TN &= \sum_{r=1}^{|X|} \min\left(f_t^2(x_r), f_g^2(x_r)\right) \\ FN &= \sum_{r=1}^{|X|} \max\left(f_t^2(x_r) - f_g^2(x_r), 0\right) \end{aligned} \tag{7.7}$$

The use of the t-norm in the definition allows the particular property

$$TP + FP + TN + FN = |X| \tag{7.8}$$

For the avoidance of doubt in the next sections, until otherwise specified, we will always use the *TP, FP, TN, FN* acronyms to indicate the *crisp* versions of the metrics in 7.5.

7.2.3 Overlap Based Metrics

We have already encountered the first two metrics back in chapter 1.3.3: the *Dice Coefficient* and the *Jaccard Index*.

The *Dice Coefficient*, also called *Overlap Index*, is the most common segmentation metric and can be redefined in terms of the *confusion matrix* as

$$DICE = \frac{2|S_g^1 \cap S_t^1|}{|S_g^1| + |S_t^1|} = \frac{2TP}{2TP + FP + FN} \tag{7.9}$$

The *Jaccard Index* can also be redefined as

$$JAC = \frac{|S_g^1 \cap S_t^1|}{|S_g^1 \cup S_t^1|} = \frac{TP}{TP + FP + FN} \tag{7.10}$$

We also notice that both the *Jaccard Index* and the *Dice Coefficient* offer the same informative content as one can be expressed in terms of the other

$$JAC = \frac{|S_g^1 \cap S_t^1|}{|S_g^1 \cup S_t^1|} = \frac{2|S_g^1 \cap S_t^1|}{2\left(|S_g^1| + |S_t^1| - |S_g^1 \cap S_t^1|\right)} = \frac{DIE}{2 - DICE} \tag{7.11}$$

or, analogously

$$DICE = \frac{2JAC}{1 + JAC} \tag{7.12}$$

The next metric is the *True Positive Rate*, also known as *Recall* or *Sensitivity*, which intuitively indicates the proportion of actual positives among every positively classified voxels. It is symmetrically opposed to the *True Negative Rate* or *Specificity*, which measures the proportion of actual background voxels among all the voxels that are negatively classified.

$$\begin{aligned} \text{Recall} = \text{Sensitivity} = TPR &= \frac{TP}{TP + FN} \\ \text{Specificity} = TNR &= \frac{TN}{TN + FP} \end{aligned} \tag{7.13}$$

The two measures that are complementarily defined are the *False Positive Rate* or *Fallout* and the *False Negative Rate*:

$$\begin{aligned} \text{Fallout} = FPR &= \frac{FP}{FP + TN} = 1 - TNR \\ FNR &= \frac{FN}{FN + TP} = 1 - TPR \end{aligned} \tag{7.14}$$

One last important metric is the *Positive Predictive Value* or *Precision*, which measures the ratio of true positives over all the positive calls.

$$\text{Precision} = PPV = \frac{TP}{TP + FP} \tag{7.15}$$

7.2.4 Receiver Operator Characteristic and Precision-Recall Curves

When deploying a segmentation model it's extremely unlikely that the outputs in the application case are in the *fuzzy* format defined in 7.2.1, what happens instead is that the model outputs binary predictions (*crisp* segmentations as defined in 7.2.1) by applying a simple threshold or a series of more advanced post-processing

operations [1]. These operations depend on additional parameters - e.g. a thresholding value - that influence the final output and for each of these values a set of different confusion matrix is produced.

A major insight on how the model performs when changing its classification parameters is given by the *Receiver Operator Characteristic Curve* (*ROC*) and the *Precision-Recall Curve*.

ROC Curve

The *ROC* curve is defined as the *True Positive Rate* values plotted against the *False Positive Rate* and serves as a device for measuring the model's ability to separate classes. Looking at definitions 7.14 and 7.13 one can easily see how, by fixing the total number of positives and negatives, specifying a point in the *TPR-FPR* space determines an unique confusion matrix so that tracing a *ROC* curve actually offers a general representation of the behaviour of the confusion matrix at the changing of the classification threshold. The area under the *ROC* curve can be used as a measure of class separability.

We introduce some graphics to better understand the role of a ROC curve: let's imagine that positives and negatives are modeled by two partially overlapping distributions and that the decision rule is represented by a threshold. An ideal model would produce two perfectly separated distributions so that, moving the threshold, for every possible false positive rate, the true positive rate would always be one as no false positives are ever registered. The ROC curve would then be identically 1, as in figure 7.5.

A completely indecisive model like the one in figure 7.6, on the other hand, would represent the two distributions as completely overlapped: in that case the TPR would increase linearly with the FPR and would result in a linear ROC curve.

Figure 7.8 shows an intermediate case in which the TPR rises overlinearly with the FPR. Best performing models - with minimal distribution overlap - will occupy the upper left parts of the FPR-TPR space.

[1] e.g. when identifying extended objects, simple thresholding may produce prediction inconsistencies that one might want to fix with morpohlogical operations. Alternatively if threhsolding produce too much prediction inconsistency one might replace it with more advanced statistical blob recognition algorithms.

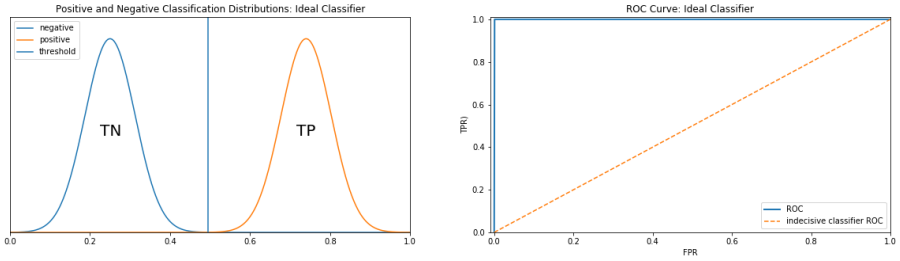

Figure 7.5: ROC Curve : Ideal Classifier

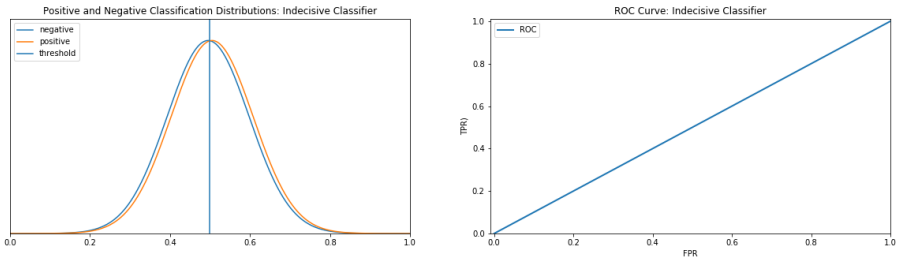

Figure 7.6: ROC Curve : Indecisive Classifier

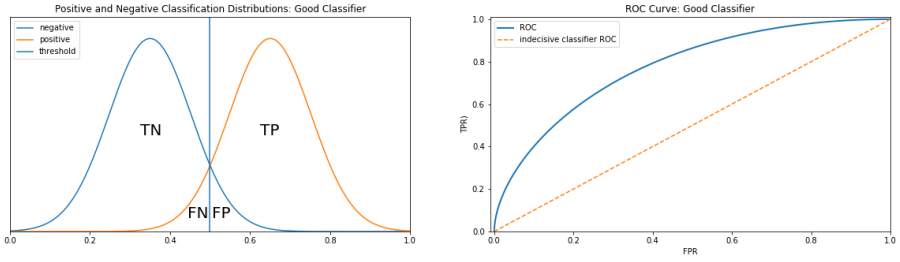

Figure 7.7: ROC Curve : Good Classifier

Underlinear ROC curves actually represent the unlucky case in which the model reciprocates the classes, systematically classifying positives as negative and negatives as positive.

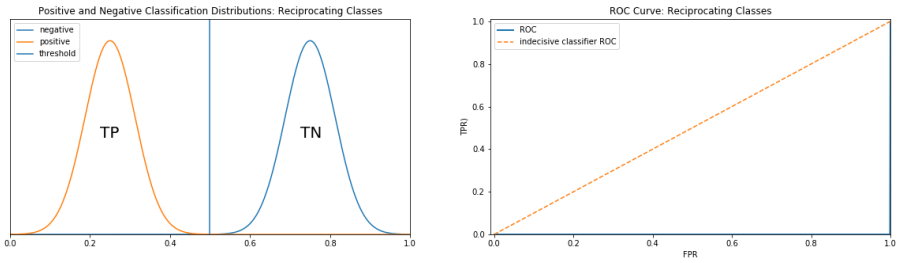

Figure 7.8: ROC Curve : Reciprocating Classes

Precision-Recall Curve

One of the drawbacks of using ROC curves is that they are prone to offering overly optimistic views of performances if the dataset suffers from class imbalance and one of the two classes is over-represented [30]: this is actually the case when trying to identify sparse objects on a background. *Precision-Recall Curves* are often used when the negative classifications largely outnumber the positive ones. Looking at definitions 7.15 and 7.13 one can see the reason of this: in ROC (TPR vs. FPR) a large number of false positives might lead to a small change in the total Fallout, while Precision offers a comparison between true positives and false positives that is less sensitive to the large number of correctly classified background instances.

If Recall is not null, there exists a one-on-one correspondence between curves in ROC space and curves in PR space, such that the same exact confusion matrices are contained in the curves: this is a direct consequence of the fact that each point in the ROC space uniquely defines a confusion matrix (if the total numbers of positives and negatives are fixed) and TP, FN, FP can be used to determine a point in the PR space. These three numbers cannot possibly represent another confusion matrix with a different TN because of the total negatives constraint. Same reasoning goes for translating a point from the PR space to the ROC space, with the only exception made by the Recall $= 0$ axis: in this case FP cannot be recovered and there is no one-on-one mapping.
We can translate point by point any curve from the ROC space to a curve in the PR space when Recall $\neq 0$: an interesting consequence of this is the fact that a curve dominates another curve in the ROC space if and only if the first dominates the other in PR space [27]. What this means is that we can equivalently compare results

in one space and be sure that the same comparison would lead to same results in the other space: this property comes in particularly handy when dealing with unbalanced datasets, which produce difficult-to-read ROC curves.

The typical aspect of a PR curve is delineated in figure 7.9: in this case the indecisive classifier produces a constant line whose value is determined by the proportion between positive and negative cases

$$P_{\text{limit}} = \frac{P}{P+N} \tag{7.16}$$

an ideal classifier would produce a constant 1 line, except for Recall = 1 where it would assume P_{limit} value. Every other classifier would trace a curve in between these two extremes.

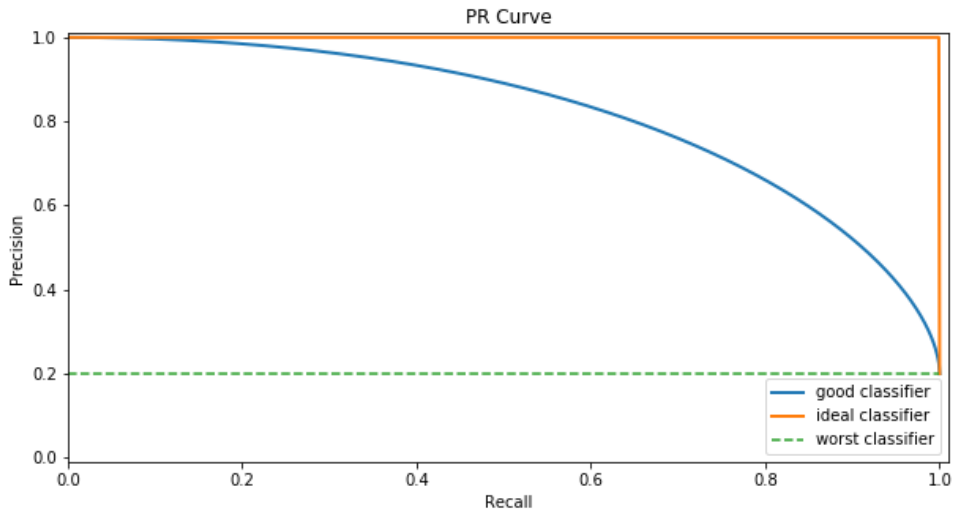

Figure 7.9: PR Curve

7.3 Electron Microscopy Dataset Analysis

Both UNET and 3D UNET styled models such as those described in section 6, with residual structural variations (at least in the 3D case), were trained on the EM dataset.

The two volumes were split in a 3-way mode as a *training* subvolume, a *validation* and a *test* one in $60:20:20$ proportions. The 2D models were trained on a 64×64 crops of the volume slices while the 3D models used random $64 \times 64 \times 64$ volumetric crops. All the models were written in *Tensorflow 1.15* using the *Keras* functional library and trained locally on a machine with an AMD 2700X CPU and an RTX2070 GPU.

In this configuration each model version was trained for 250 epochs, where the epoch size was decided by estimating the number of passes over the training data:

$$\begin{aligned} \text{epoch length}_{2D} &= \text{ceil}\Big(\frac{t_x t_y t_z}{w_x w_y}\Big) \\ \text{epoch length}_{3D} &= \text{ceil}\Big(\frac{t_x t_y t_z}{w_x w_y w_z}\Big) \end{aligned} \tag{7.17}$$

where t_x, t_y, t_z and w_x, w_y, w_z are respectively the dimensions of the training volume and the input window. The total training area consisted of a $1024 \times 768 \times 165$ volume, each training session took a computation time of approximately 5 hours [2]

7.3.1 Model Comparison

Model Name	***AUC-ROC***	***AUC-PR***	***DICE***	***JAC***
UNET3D	0.9903 ± 0.0001	0.918 ± 0.001	0.84 ± 0.01	0.73 ± 0.01
Residual 3D U-NET: Post-Act	0.9890 ± 0.0002	0.961 ± 0.001	0.91 ± 0.01	0.83 ± 0.01
Residual 3D U-NET: Pre-Act	**0.9957 ± 0.0001**	**0.967 ± 0.001**	**0.92 ± 0.01**	**0.85 ± 0.01**
UNET	0.9951 ± 0.0001	0.938 ± 0.001	0.86 ± 0.01	0.75 ± 0.01
Residual U-NET: Post-Act	0.9955 ± 0.0001	0.940 ± 0.001	0.86 ± 0.02	0.76 ± 0.03
Residual U-NET: Pre-Act	**0.9957 ± 0.0001**	0.951 ± 0.001	0.88 ± 0.02	0.78 ± 0.03

Model Name	***FALLOUT***	***FNR***	***PREC.***	***RECALL***	***SPEC.***
UNET3D	0.010 ± 0.002	0.12 ± 0.02	0.81 ± 0.03	0.88 ± 0.02	0.989 ± 0.002
Residual 3D U-NET: Post-Act	0.004 ± 0.001	0.09 ± 0.01	**0.91 ± 0.01**	0.91 ± 0.01	**0.996 ± 0.001**
Residual 3D U-NET: Pre-Act	0.004 ± 0.001	**0.08 ± 0.01**	**0.91 ± 0.01**	**0.92 ± 0.01**	**0.996 ± 0.001**
UNET	**0.003 ± 0.001**	0.19 ± 0.03	**0.91 ± 0.02**	0.81 ± 0.03	**0.996 ± 0.001**
Residual U-NET: Post-Act	0.005 ± 0.001	0.16 ± 0.02	0.88 ± 0.02	0.84 ± 0.02	0.994 ± 0.001
Residual U-NET: Pre-Act	0.004 ± 0.001	0.14 ± 0.02	0.90 ± 0.02	0.86 ± 0.02	0.996 ± 0.001

Predictive performance was evaluated on the test dataset partition using the metrics in 7.2: both models use the same training, testing and validation partitions and

[2] Possibly better training times could have been reached using more efficient data augmentation methods.

the 2D model performances are evaluated over the entire 2.5D-reconstructed testing volume. Values in the summary table are obtained by setting a 0.5 probabilistic threshold (every response over 0.5 is considered positive)[3], Receiver Operator Characteristic and Precision Recall curves are obtained by setting different thresholds and serially evaluating true positive ratio, false positive rate and positive predictive values for each value.

[3]Confidence intervals of the various metrics and curve integrals, both in this case and in the fluorescence microscopy one, are clearly under-estimated. An accurate K-fold or bootstrap analysis of the errors would have required the study of a whole new training strategy that support the use of uncontiguous volumes: a complete redesign of more efficient and flexible training data generators is in the near future goals of my PhD. We chose to use a differentiated error evaluation strategy for AUCs and static metrics: for ROC and PR areas we used a bi-normal approximation [11] [61] to estimate the expected variances, obtaining errors that are roughly compatible with the observed comparison points, but following the same approximation for the other metrics revealed itself to be an unfruitful path as the resulting expected variances were way under the observed comparison points. We chose to use as a comparative tool, in this case, a standard error evaluated over ten different takes of the metrics in a sphere of 0.1 around the actual decision threshold. It's also to be noted that asymptotically low errors in ROC curve integrals (and general readability of the curves themselves) are to be ascribed to very strong class imbalance (in FM datasets positive voxels are less than 4% of the total): ROC curves suffer the fact that the almost totality of the correctly classified examples are background voxels. In this case the area under PR curves should offer a more representative datum but, being limited to [0,1], high values of AUC-PR show again asymptotical effects.

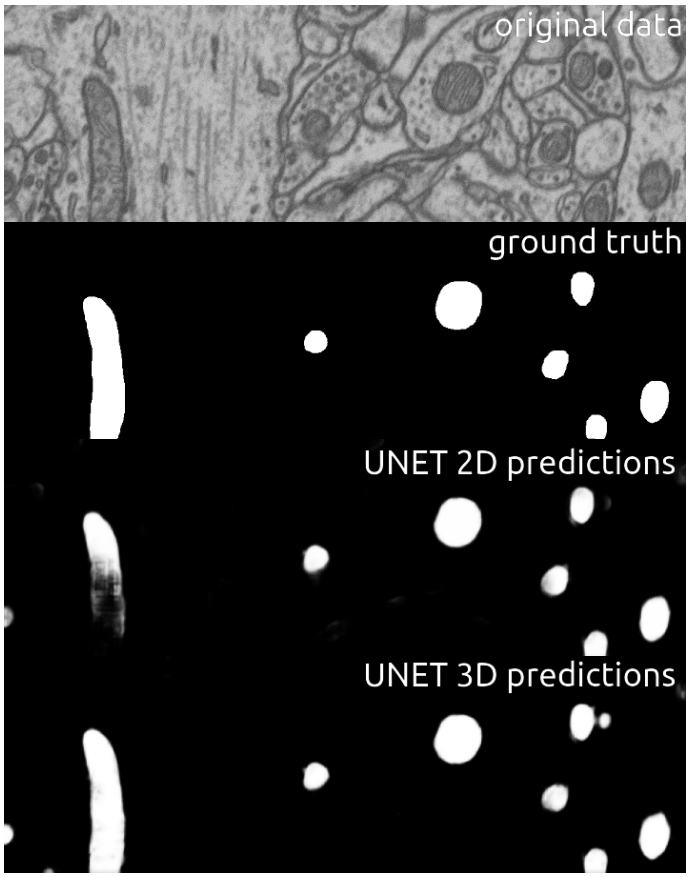

Figure 7.10: EM Dataset: Training Data, GT, 2D and 3D Predictions

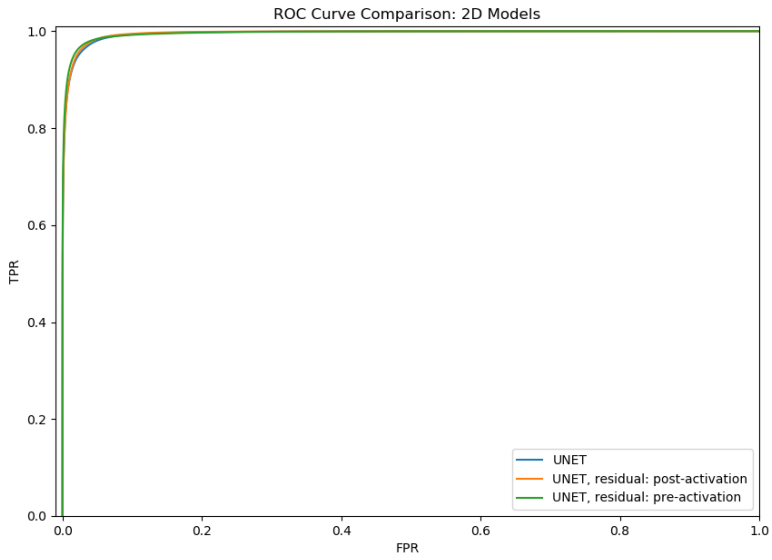

Figure 7.11: EM Dataset, ROC Curve Comparison: 2D Models

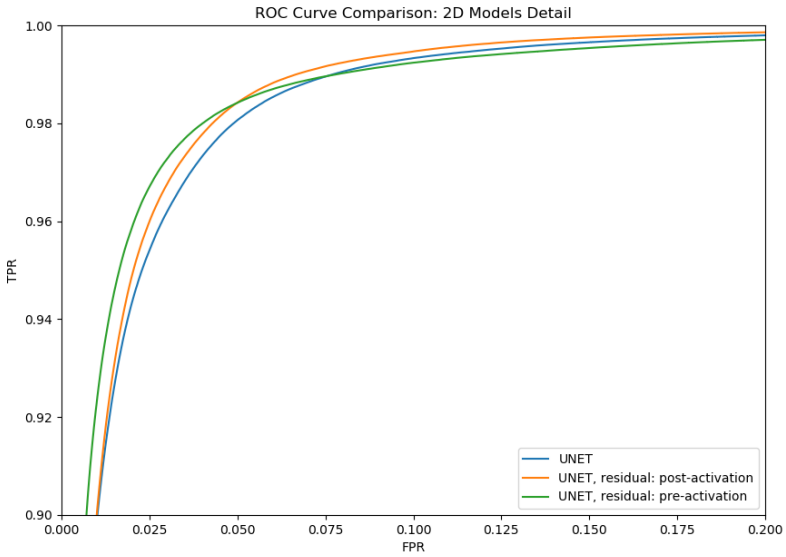

Figure 7.12: EM Dataset, ROC Curve Comparison: 2D Models, Detail

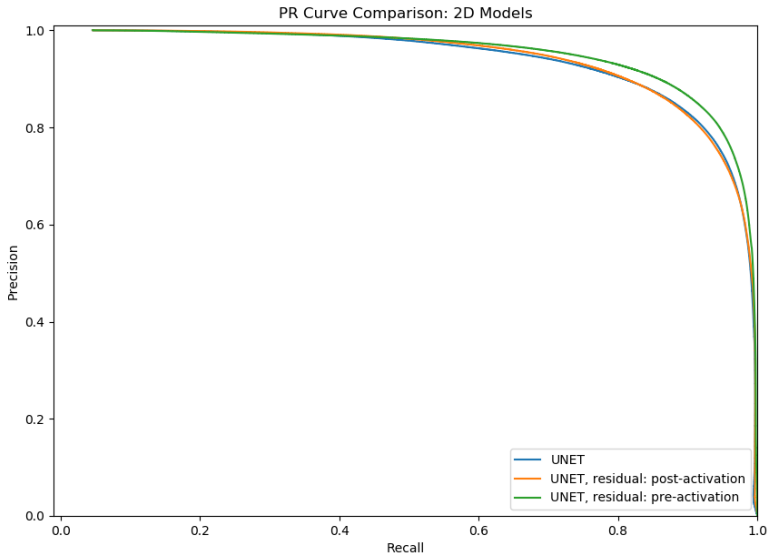

Figure 7.13: EM Dataset, PR Curve Comparison: 2D Models

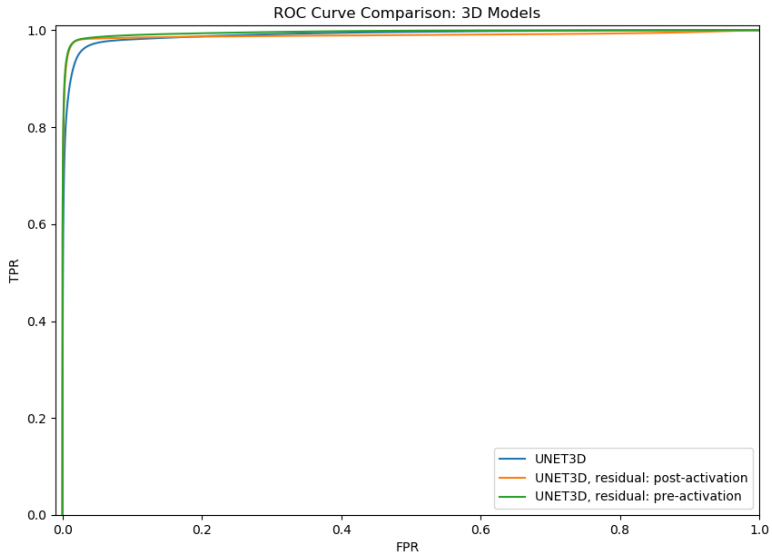

Figure 7.14: EM Dataset, ROC Curve Comparison: 3D Models

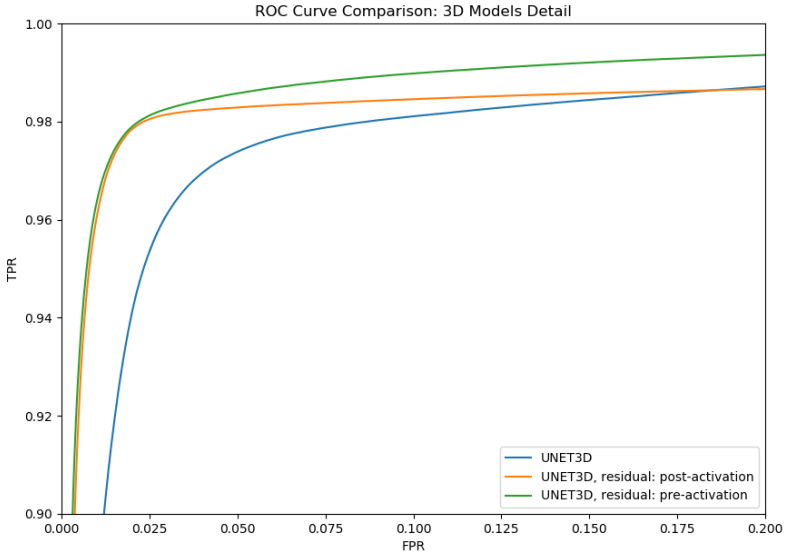

Figure 7.15: EM Dataset, ROC Curve Comparison: 2D Models, Detail

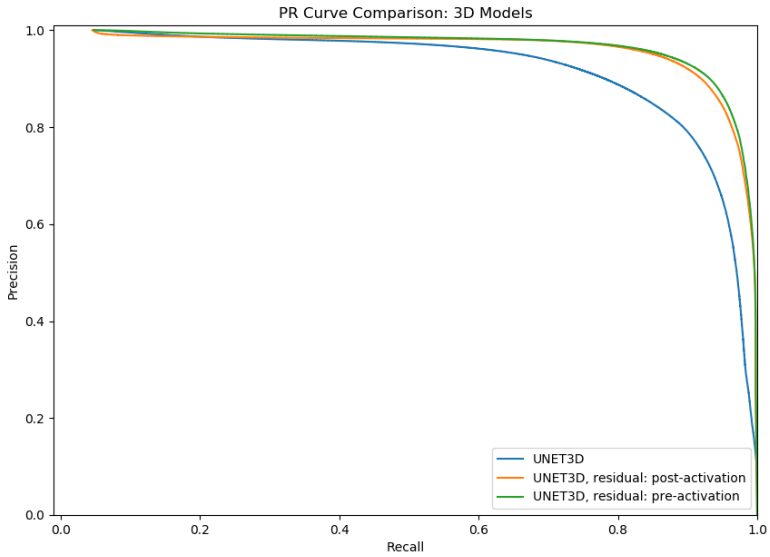

Figure 7.16: EM Dataset, PR Curve Comparison: 3D Models

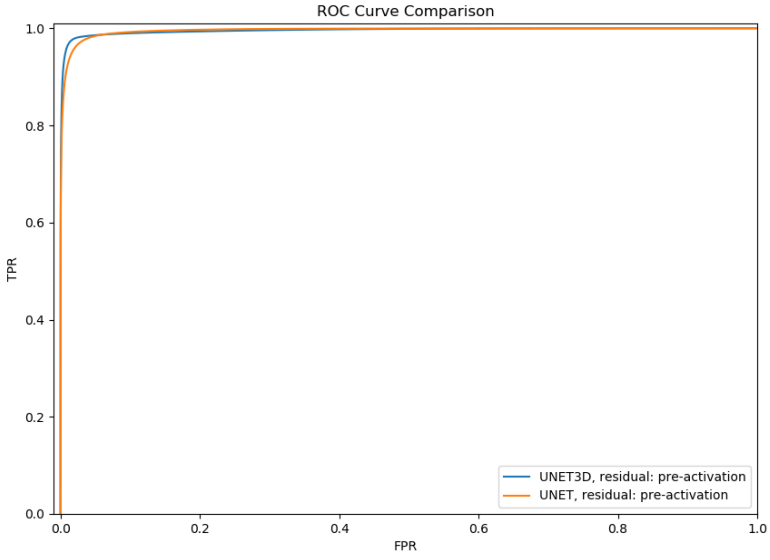

Figure 7.17: EM Dataset, ROC Curve Comparison: 2D vs 3D

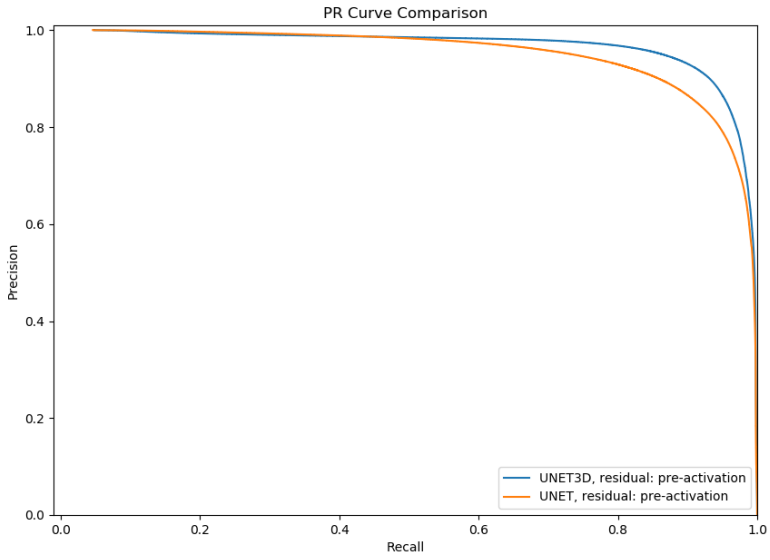

Figure 7.18: EM Dataset, PR Curve Comparison: 2D vs 3D

A comparison is made among six different models: three 2D UNET-syle models and three different 3DUNET models. For each of the groupps, the first is based on on, respectively, the original U-NET paper [99] by Ronneberger and 3D-UNET model by Cicek [133], the other two are, respectively a naïve implementation of residual learning in the first model, realized by inserting a skip connection (1×1 convolution + batch normalization) between the inputs and the outputs of the convolutional blocks, and the other is a pre-activation setup that replaces the convolutional blocks with the residual blocks described in figure 6.7. The high number of negative cases in the dataset makes the ROC curves in figure 7.11 and 7.14 difficult to read, we provide a detail of the upper-left quadrant in figures 7.12 and 7.15 in which the single curves are more recognizable. The Precision-Recall curve in figures 7.13 and 7.16 should be readable regardless of class imbalance.

Starting from 2D model comparison, both figure 7.11 and 7.13 highlight a slight preference for the residual model implementations, with marginal differences in the area integrals, the same is to be observed when comparing 3D models.

The comparison between the best 2D model and the best 3D model reveal a

noticeable (at least in the PR curve) performance improvement of the residual implementation of 3D U-NET over the residual implementation of U-NET.

It's to be stressed out that the models were trained on the same amount of data. The number of *true* examples the 3D model is exposed to is geometrically lower than the number of examples 2D models can train upon. Looking at expression 7.17 it's easy to see that the 2D model receives w_z times the examples of the 3D ones: when considering $64 \times 64 \times 64$ windows, there's a factor of 64 between the amount of employed examples in the 2D and 3D case.
The performances of 3D models can reasonably be expected to increase when the same numbers of training examples are used. In this setup we preferred to make comparisons keeping the same amount of original data to give a sense of what the expected performances when a plausible scenario, in which the 3D models are used as a drop-in replacements for the 2D ones, is considered.

Looking at the metric summary table, the so underpowered Residual 3D UNET with full pre-activation residual blocks still seems to achieve best results both on ROC and PR integral areas and "static" metrics (decision threshold set to 0.5), with exception made for the False Positive Rate, Precision, True Positive Rate and True Negative rates that are compared beyond the third decimal digit, resulting in practically identical performances between best achievers in those areas.

7.3.2 Surface Reconstruction

Three-dimensional surface representations of predictive results were obtained using the *marching cubes* algorithm described in section 6.5, generating meshes in the standard `.stl` format to be readable by any 3D manipulation program like *Blender*. Although no meaningful quantitative comparison between the meshes generated by different models was carried out, we wanted to make a few qualitative remarks on the appearance of reconstructed objects.

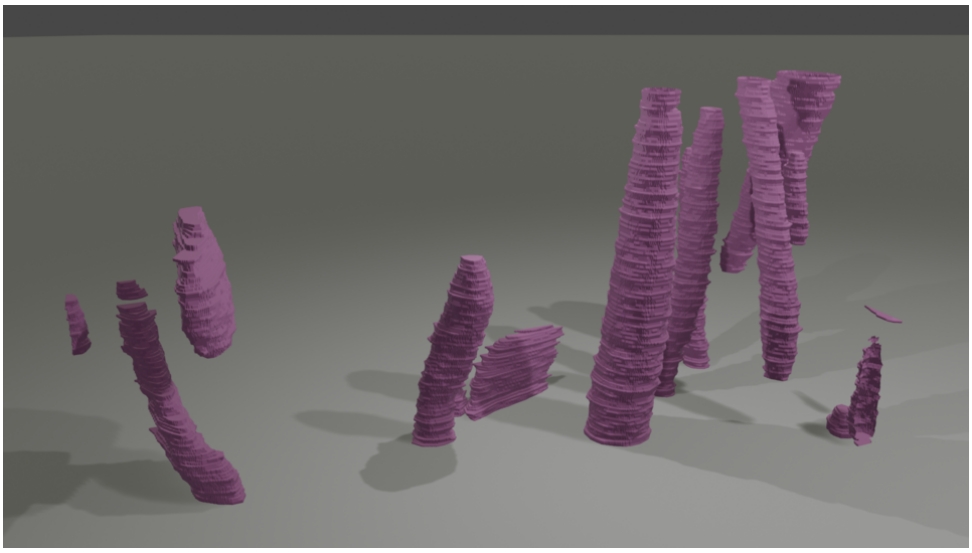

Figure 7.19: EM Dataset: Surface Reconstruction from GT

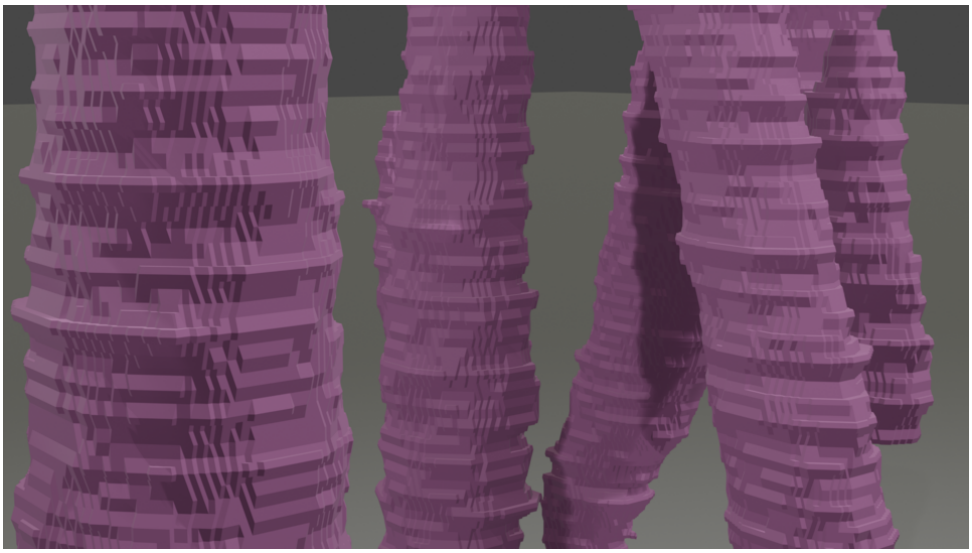

Figure 7.20: EM Dataset: Surface Reconstruction from GT: Detail

In figure 7.19 and 7.20 we can see the surface reconstructed from baseline groundtruth segmentation of the data. A rapid visual inspection reveals a blocky and jagged appearance of the surfaces that can be explained by two order of reasons: in the first place the 2D slice-by-slice manual segmentation method inevitably leads to minor

inconsistencies between subsequent appearances of the same object as the manual operator has to sequentially trace the same objects in consecutive volume slices. In secundis, a less trivial cause for jagged appearance of said surfaces can be pointed back to how the Marching Cubes algorithm works. Marching Cubes creates isosurfaces of a scalar field sampled over a 3D lattice by evaluating, for every set of 8 contiguous lattice points, what is the general topology class of the isosurface intersection with the faces of the cube they describe, among 256 different possibilities. This is done by assigning a binary label to each vertex depending on whether its value is above or below a certain threshold (corresponding to the wanted isovalue) and using a reference table to a particular geometry for each of the 8 bit combinations. The actual position of each of the crossing points is determined using bilinear interpolation between the original values of the vertices relative to the interested edge. What happens if we feed the Marching Cubes algorithm a binary image, like the ground truth one, is that each of the bilinear interpolation evaluations is going to give exactly 0.5 as result, meaning that each found isosurface intersection point will be placed exactly halfway on its edge. The results are not in any way less accurate than ones obtained using continuous valued images and Marching Cubes gives a coherent representation of the isosurface at a voxel resolution level. The only noticeable difference is in the subvoxel estimation of the isosurface which appears more blocky in the case of a binary image. The following figures show the surface reconstructions made from predictions of both 3D and 2D models

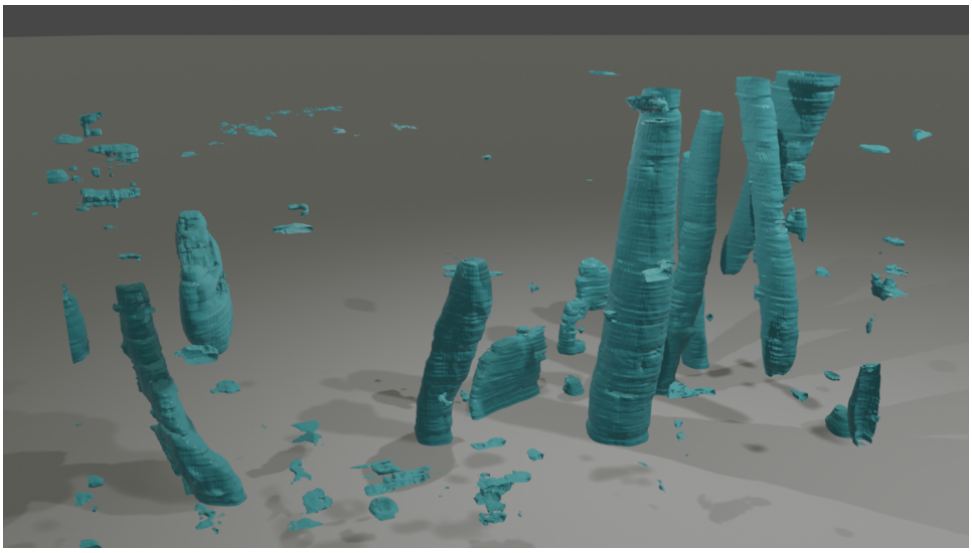

Figure 7.21: EM Dataset: Surface Reconstruction from 2D Predictions

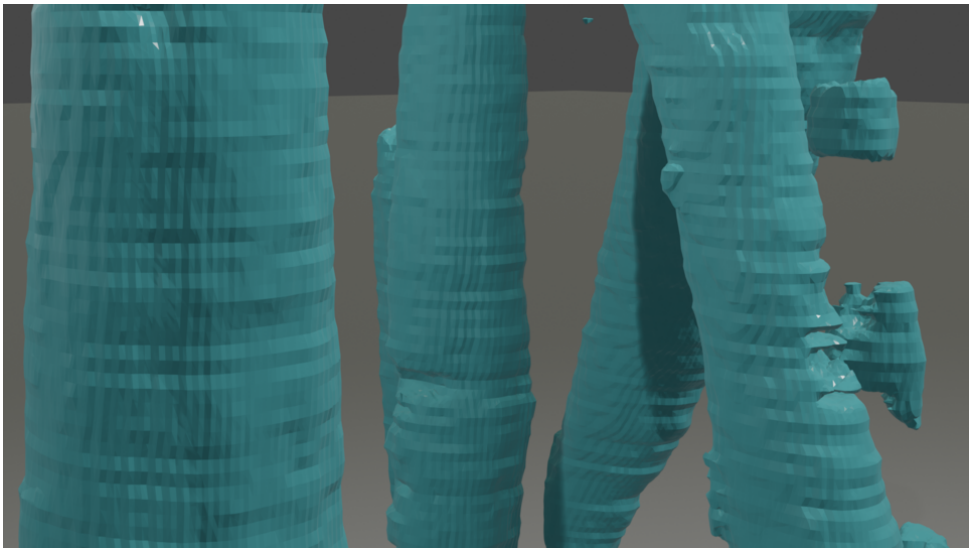

Figure 7.22: EM Dataset: Surface Reconstruction from 2D Predictions: Detail

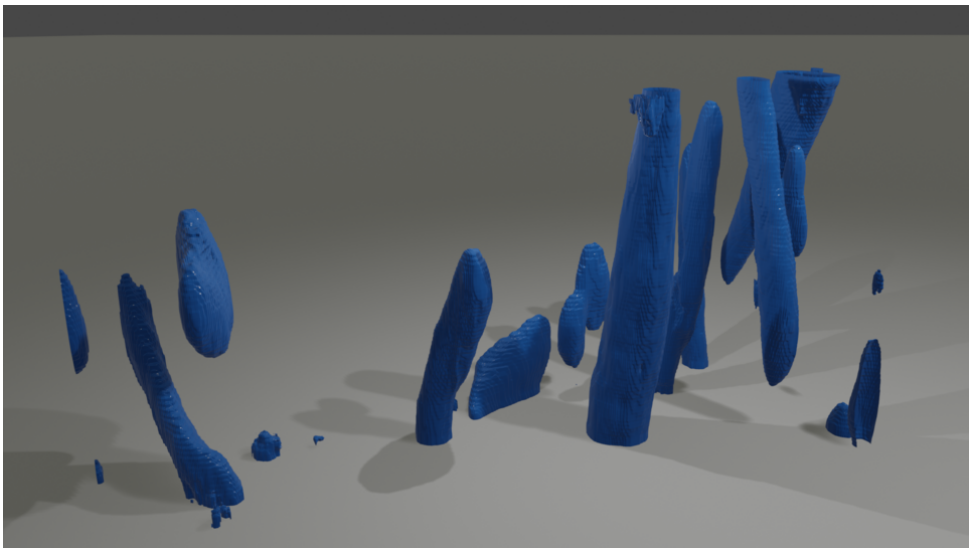

Figure 7.23: EM Dataset: Surface Reconstruction from 3D Predictions

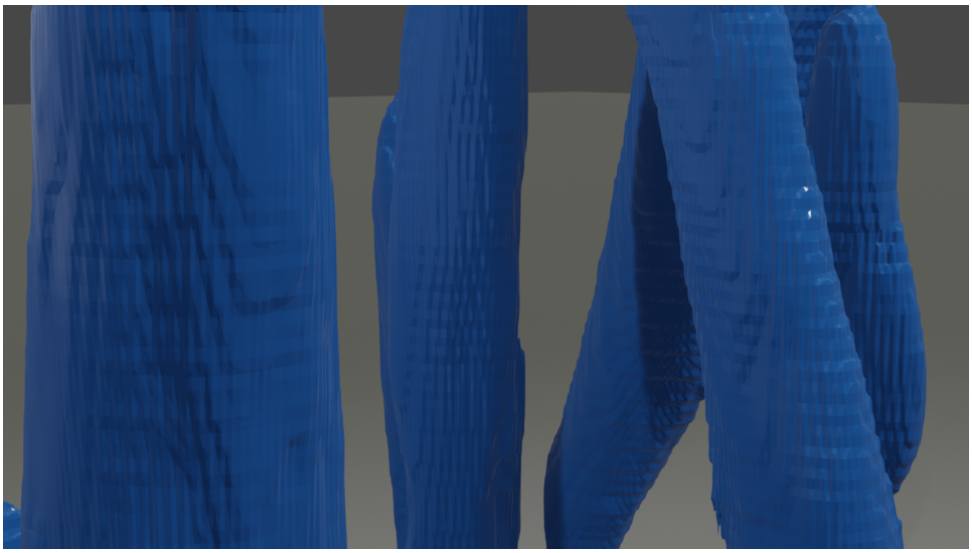

Figure 7.24: EM Dataset: Surface Reconstruction from 3D Predictions: Detail

The first thing we notice is an overall smoothness of the surfaces in figures 7.21 and 7.23 when compared to 7.19 that is to tribute at least partly to the subvoxel interpolation effects discussed above. Comparing the reconstructions from 2D model

predictions and 3D model predictions, where in both cases a 0.5 classification threshold was used to find the isosurfaces, we can point out a that the 3D model produces a globally more readable image with less false positives. Looking at details 7.22 and 7.24, with particular attention to the rightmost object, we can have a visual confirmation of increased coherence in the z-axis as in the 3D case the objects tend not to show discontinuities: this is one of the positive effects we expected from using three-dimensional receptive fields.

7.4 Fluorescence Microscopy Dataset Analysis

Training 3DCNNs requires fully-labeled data volumes or extended stacks with fully annotated images. This kind of data is extremely difficult and time-consuming to produce and, at the current time, no large human-labeled versions of this dataset exist (even though strategies for efficient labeling of large 3D frames without geometrically increasing human segmentation time are being studied at the moment). The only kind of annotated data available for this dataset are 2D frames made using LAIRA™(*"Laira is an AI-based Research Assistant"* [66]), a proprietary tool by Bioretics S.r.L. for ML-assisted data labeling: this kind of data is not suitable for 3DCNN training, which requires entirely labeled stack. A compromise path had to be found. The adopted solution is represented by an intermediate form of annotation in which 3D volumes are "pre-labeled" using a pre-existing 2D classifier, trained on 2D sparse data, to densely annotate 3D labels one slice at a time: the so obtained 3D extended pseudo-labels are then used to train the 3D model.

The following scheme represents the actual course of action we followed: a 2DCNN model recognizing neuron somas in 2D slices of TPFM images [80] [2] was trained on 2D sparsely annotated data using Aliquis™, a proprietary MachineVision framework by Bioretics srl, and was used to serially produce segmentation maps of every slice of an extended volume. These maps served as a pre-annotation for another volume which I proceeded to manually refine in the most critical cases: the results of this kind of annotation are obviously suboptimal as I'm not a biologist and do not have the required expertise to produce accurate segmentations: the obtained labels are to be considered noisy and inaccurate versions of a true segmentation. The 3DCNN

model was trained on this semi-automatically annotated version of the dataset, producing a model that is to be interpreted as a *weak learner*: the algorithm learns from noisy and inaccurate data and the resulting segmentation is weakly correlated to the actual belonging classes, hopefully still being able to generalize well enough to produce usable results.

Improving the performance of such weakly trained model is possible by reformulating the problem as an *ensemble learning* task where multiple models are trained on different data samples and average their responses to reduce the final classification variance. The complete reformulation of this segmentation problem into a *weak learning* problem exceeds the scope of this work -which is to be considered a starting point and a proof of concept for future more accurate applications of 3D Convolutional Neural Networks to this particular dataset- so the results of just a single model at a time are considered, expecting very low performance when compared to a model trained on properly annotated data.

The number of employed epochs is chosen using 7.17: being the training dataset significantly larger ($2244 \times 2060 \times 112$ voxels) than the Electron Microscopy Dataset training times are much longer, reaching 15 to 17 hours in some cases.[4]

7.4.1 Model Comparison

Model Name	***AUC-ROC***	***AUC-PR***	***DICE***	***JAC***
3D U-NET	0.9956 ± 0.0001	0.893 ± 0.001	0.79 ± 0.01	0.66 ± 0.01
Residual 3D U-NET: Pre-Act	0.9952 ± 0.001	0.885 ± 0.001	0.75 ± 0.03	0.60 ± 0.03
U-NET	**0.9975 ± 0.001**	**0.935 ± 0.001**	**0.85 ± 0.01**	**0.74 ± 0.01**
Residual U-NET: Post-Act	0.9975 ± 0.001	**0.935 ± 0.001**	**0.85 ± 0.01**	**0.74 ± 0.01**
Residual U-NET: Pre-Act	0.9975 ± 0.001	**0.935 ± 0.001**	0.84 ± 0.01	0.73 ± 0.02

Model Name	***FALLOUT***	***FNR***	***PREC.***	***RECALL***	***SPEC.***
3D U-NET	0.005 ± 0.001	0.23 ± 0.05	0.83 ± 0.03	0.76 ± 0.05	0.995 ± 0.001
Residual 3D U-NET: Pre-Act	**0.003 ± 0.001**	0.34 ± 0.06	0.88 ± 0.03	0.66 ± 0.06	0.996 ± 0.001
U-NET	0.004 ± 0.001	**0.16 ± 0.03**	0.87 ± 0.03	0.83 ± 0.03	0.996 ± 0.001
Residual U-NET: Post-Act	0.004 ± 0.001	**0.16± 0.03**	0.86 ± 0.02	**0.84 ± 0.03**	0.996 ± 0.001
Residual U-NET: Pre-Act	**0.003 ± 0.001**	0.19 ± 0.02	**0.89 ± 0.02**	0.80 ± 0.04	**0.997 ± 0.001**

[4] Again, efficient implementations could drastically reduce training times

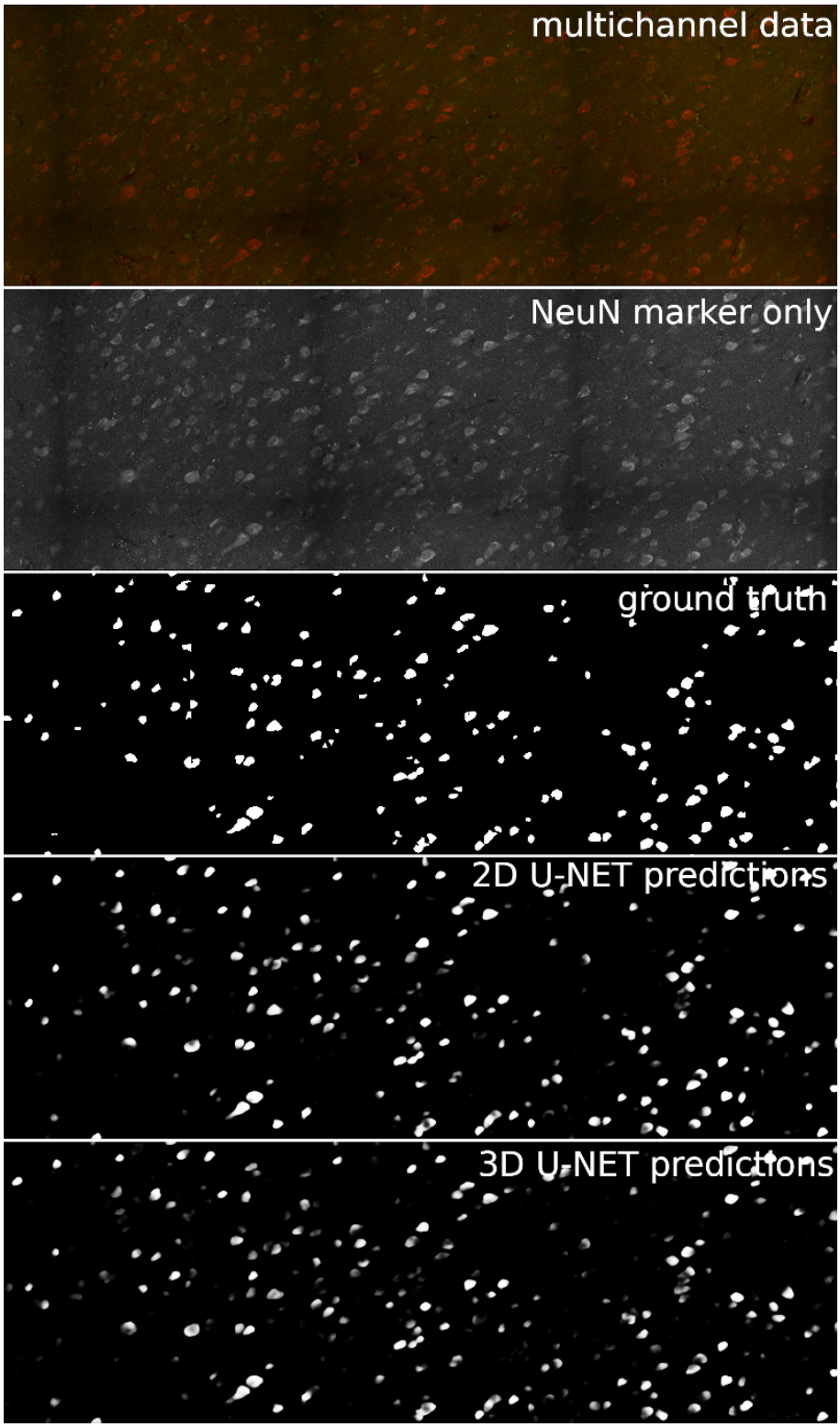

Figure 7.25: FM Dataset, Training Data, GT and Predictions

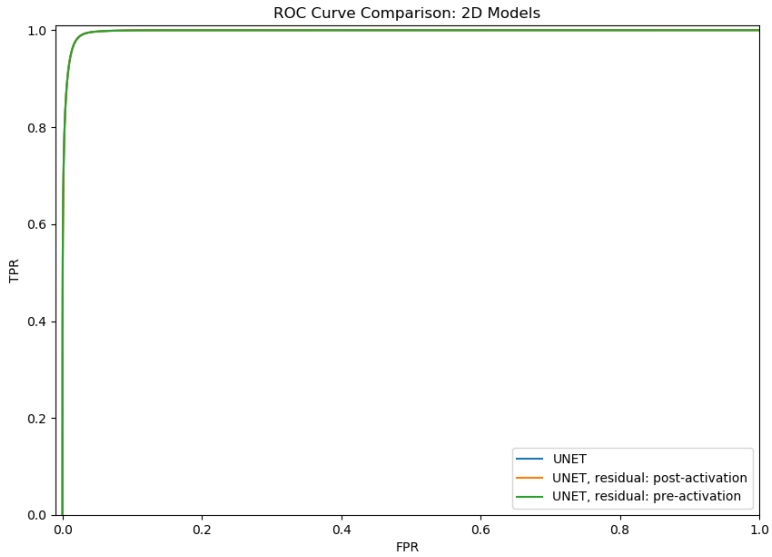

Figure 7.26: FM Dataset, ROC Curve Comparison: 2D Models

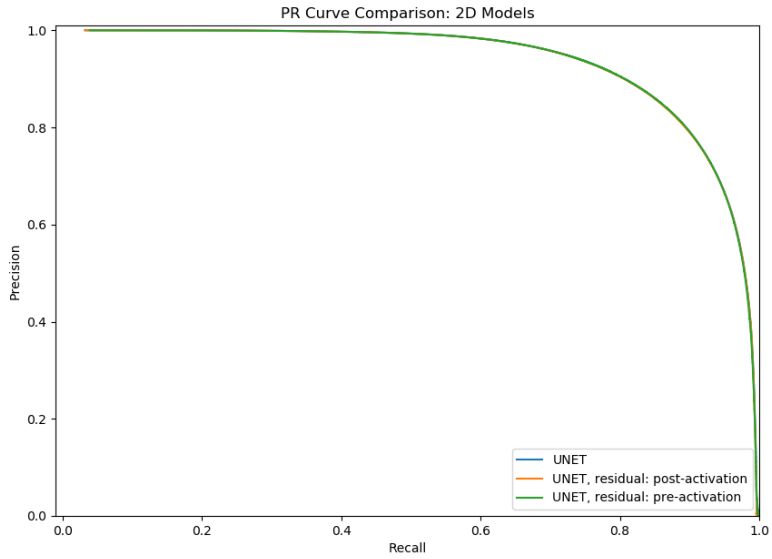

Figure 7.27: FM Dataset, PR Curve Comparison: 2D Models

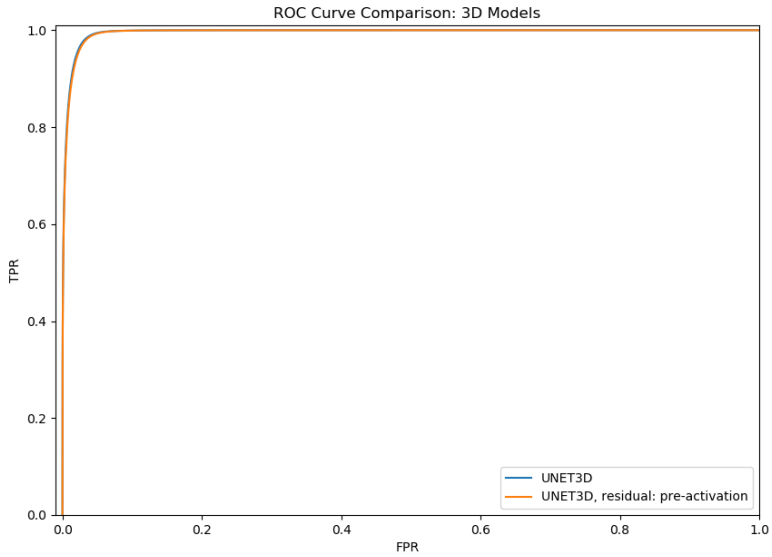

Figure 7.28: FM Dataset, ROC Curve Comparison: 3D Models

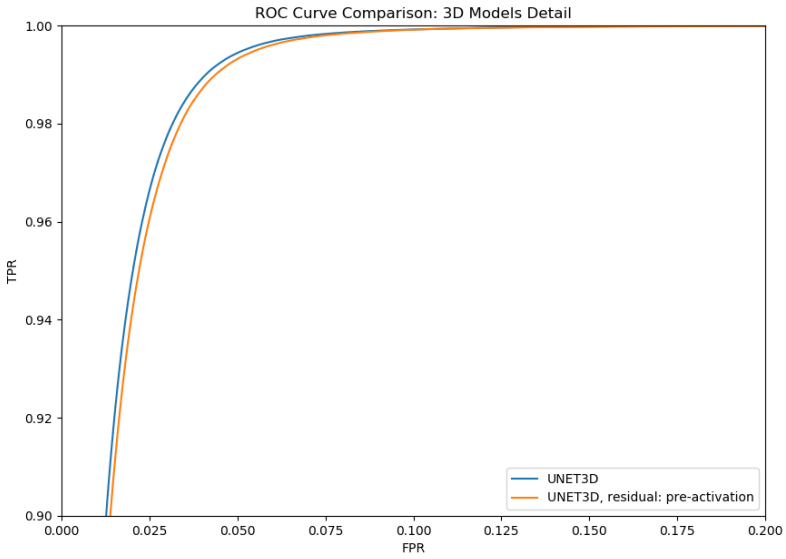

Figure 7.29: FM Dataset, ROC Curve Comparison: 2D Models, Detail

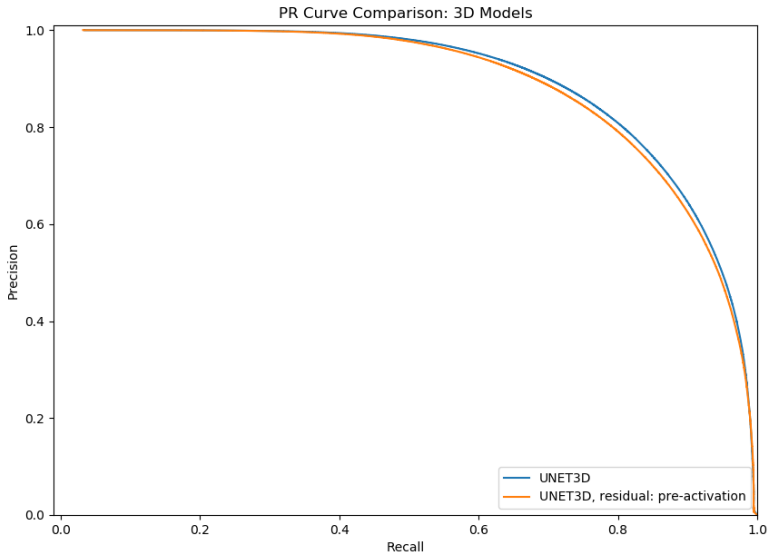

Figure 7.30: FM Dataset, PR Curve Comparison: 3D Models

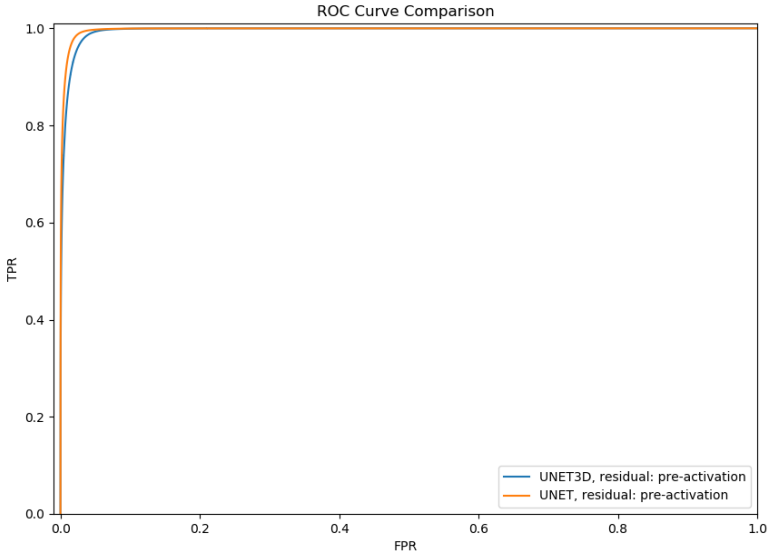

Figure 7.31: FM Dataset, ROC Curve Comparison: 2D vs 3D

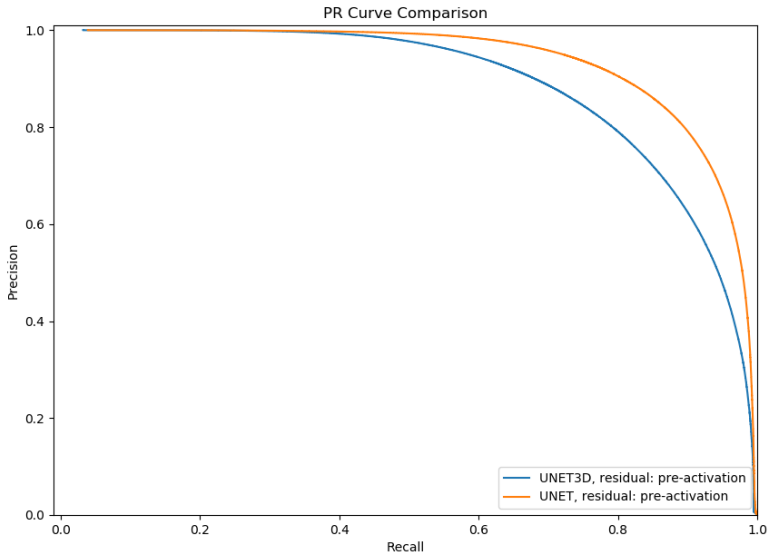

Figure 7.32: FM Dataset, PR Curve Comparison: 2D vs 3D

In this case, both ROC and PR internal comparisons between 2D implementations (non-residual, residual with post-activation and residual with pre-activation) and 3D ones in figures 7.26 and 7.28 highlight basically no difference between performances of models in the same class. A definite preference for 2D models over 3D ones is registered in PR comparison in figure 7.32.

The same emerges when looking at the summary table where 2D models apparently achieve best results in both curve integrals and most static metrics, even if a straight preference for the residual implementation of U-NET is not expressed. What's more interesting is that the residual implementation of 3D U-NET scores best in False Positive Rates, True Positive Rates and, in a more marginal way, in True Negative Rates: this implies that the model is less likely to erroneously address as positive a background pixel than the 2D one and suggests that most of the "inaccurate" classifications might come from "erroneous" positive assignments. The reason why we used quotes on *erroneous* and *inaccurate* in the last sentence comes from the observation that the reference ground truth is actually a loosely annotated version, obtained from an independent 2D model, not actually representing a *true*

annotation. Lowest Fallout rates for the 3D models are compatible with the hypothesis that a good portion of the classification error comes from marginal slices in objects that are extended in the z-stacks, that these objects might not be correctly recognized in the original segmentation (see figure 2.12), and that the 3D model generalizes enough that it tries to signal their presence in top and bottom slices, in disagreement with the adopted GT.

Unfortunately, in absence of a human-labeled dataset we cannot conclusively prove or disprove this hypothesis but a visual comparison between ground truth data and model outputs on the test data can be made. This question will hopefully be explored in depth in future works on the same theme.

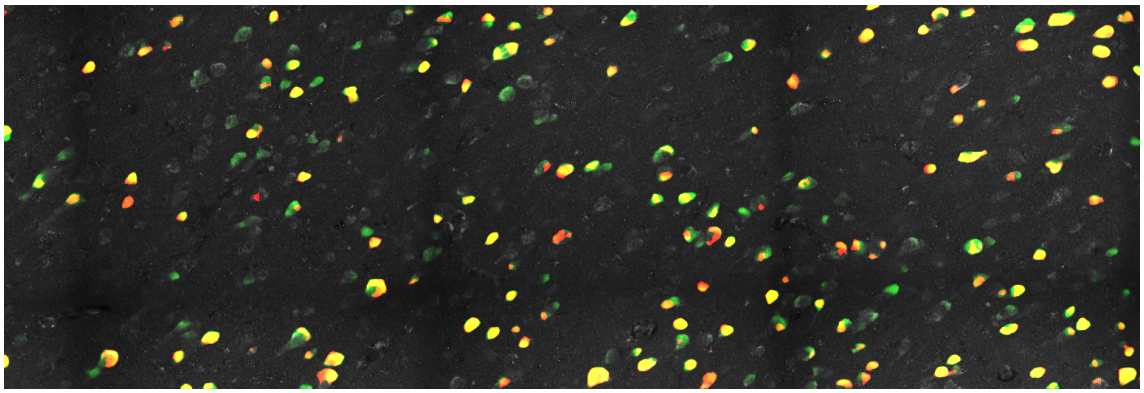

Figure 7.33: Comparison Between GT and 3D Prediction

In figure 7.33 red channel is assigned to ground truth labels and green channel is related to 3D Residual U-NET predictions (yellow pixels represent overlap between ground truth data and predictions), by exploring the stack in the z dimension we can see examples like the one in figure 7.34 where the object is not initially acknowledged by the GT but is positively predicted by the model. The same thing happens where the spatial extent of the object ends in the z-stack.

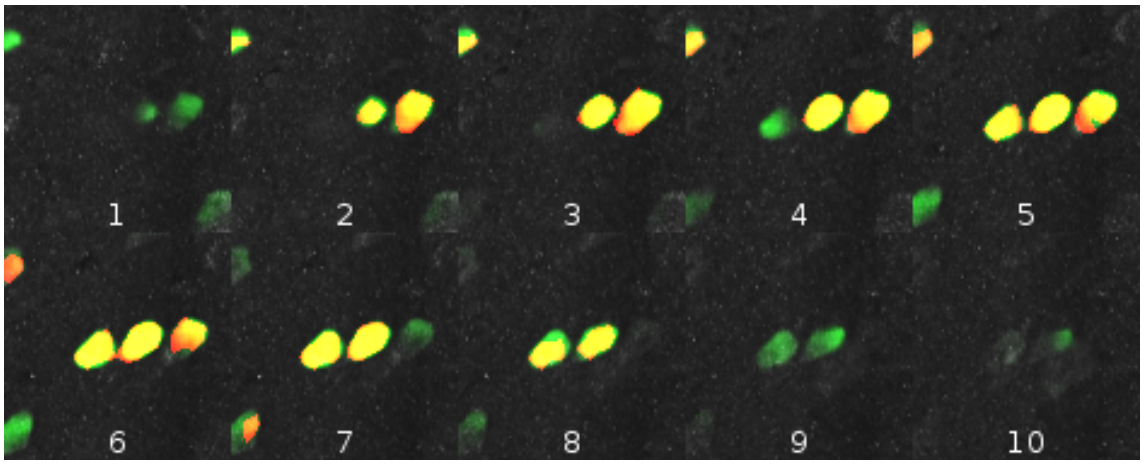

Figure 7.34: Detail of GT and Prediction Comparison: Cropping Artifacts in GT

If our explanation is confirmed with the means of a small (but expectedly expensive in terms of biologist work-hours) test dataset, this could be a strong confirm that the model successfully generalizes from unreliable ground truth data, posing the foundations for an approximate segmentation strategy that could exponentially reduce deployment costs of this kind of models.

7.4.2 Surface Reconstruction

Surface reconstruction from GT, figure 7.35, again suffers from the same blocky and jagged appearance caused by Marching Cubes applied to binary images: in this case, where the objects of interest are much smaller, image readability is noticeably degraded by this effect as we can see in figure 7.36.

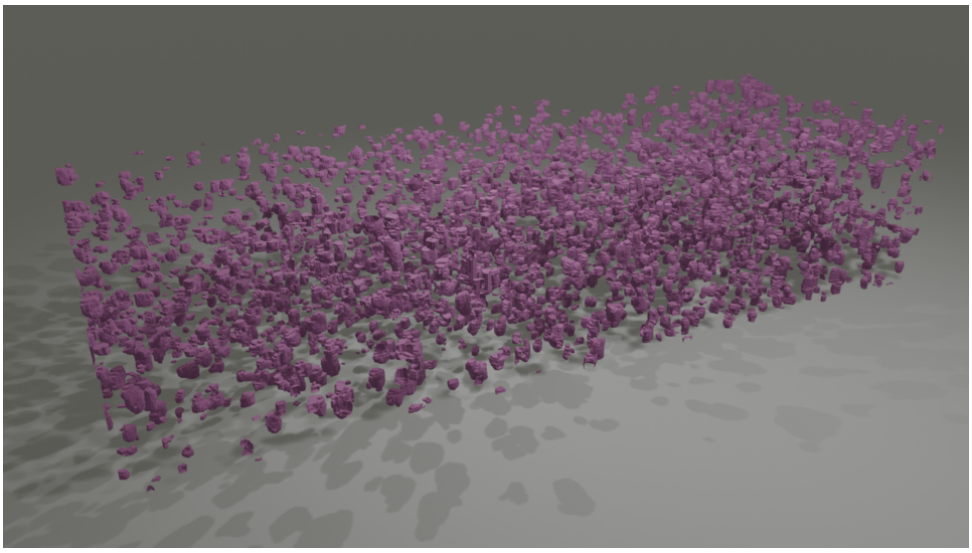

Figure 7.35: FM Dataset: Surface Reconstruction from GT

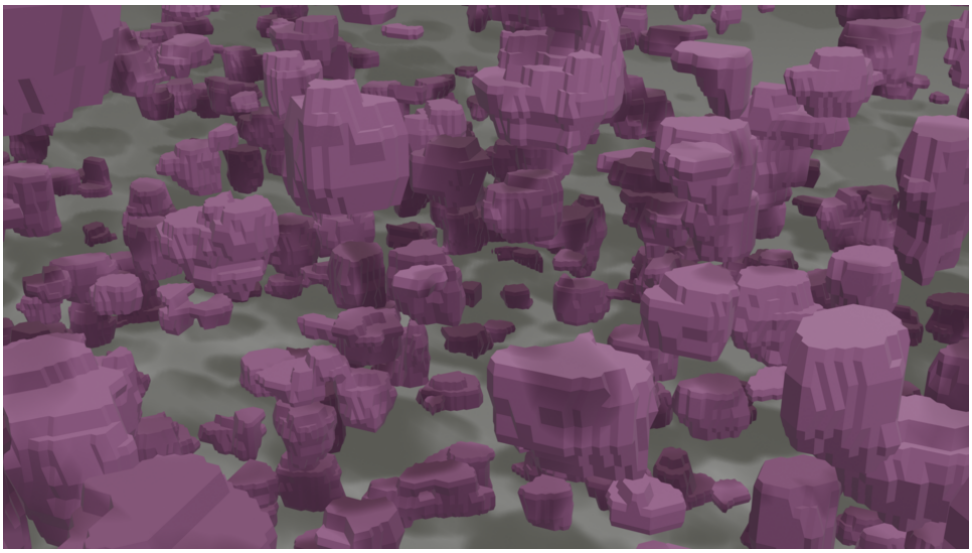

Figure 7.36: FM Dataset: Surface Reconstruction from GT: Detail

2D and 3D reconstructions, are very similar in a large-scale view (figure 7.37 and figure 7.39) but show minor visual differences at a smaller scale (figure 7.38 and 7.39) with the 2D model producing slightly more conservative segmentations, resulting in apparently bigger objects and less defined surfaces.

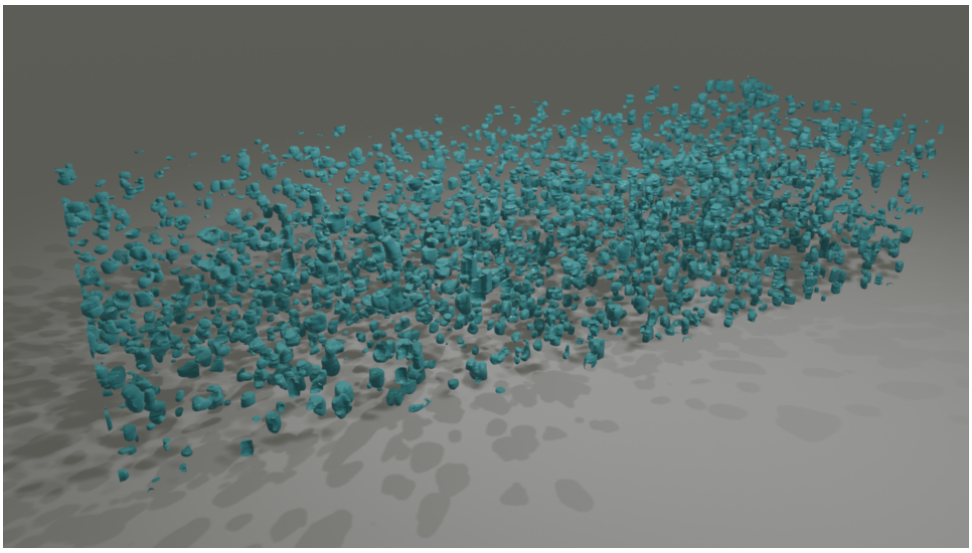

Figure 7.37: FM Dataset: Surface Reconstruction from 2D Predictions

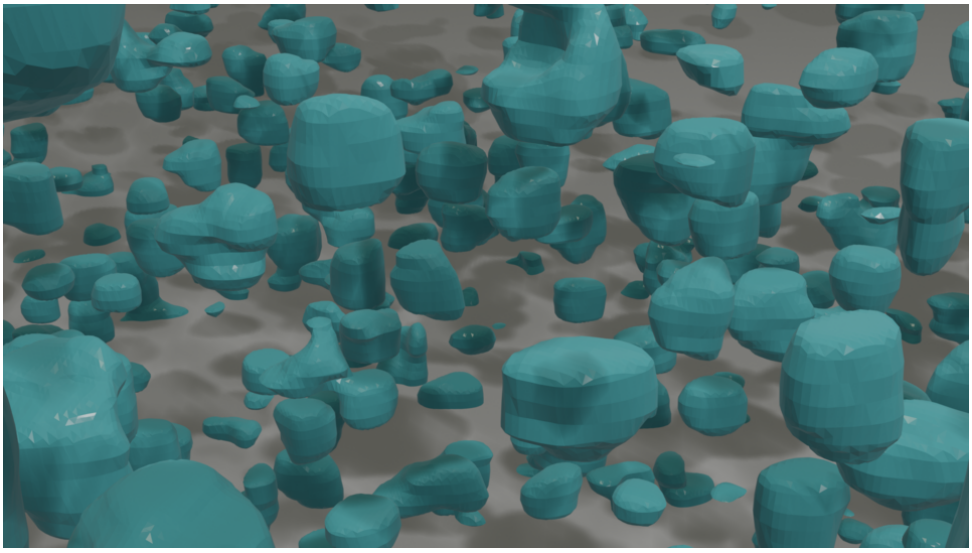

Figure 7.38: FM Dataset: Surface Reconstruction from 2D Predictions: Detail

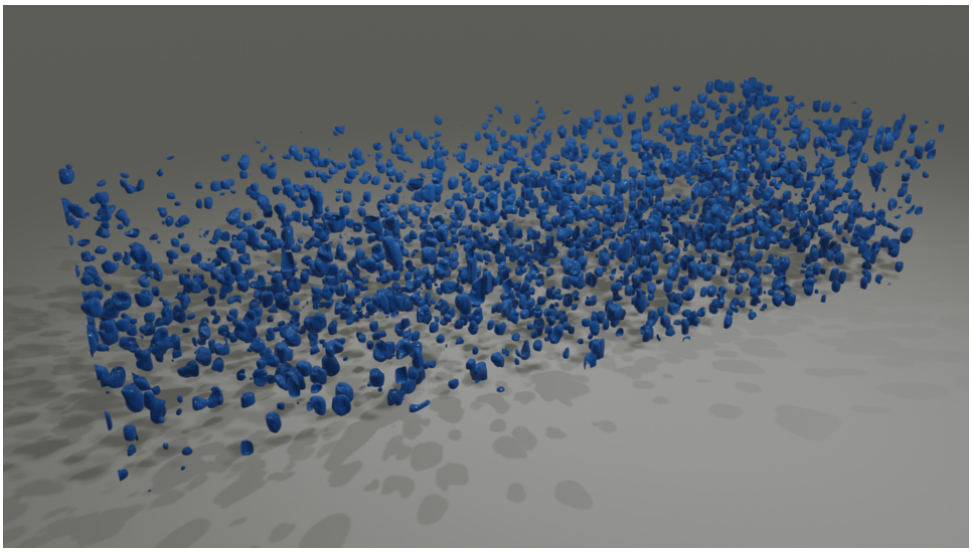

Figure 7.39: FM Dataset: Surface Reconstruction from 3D Predictions

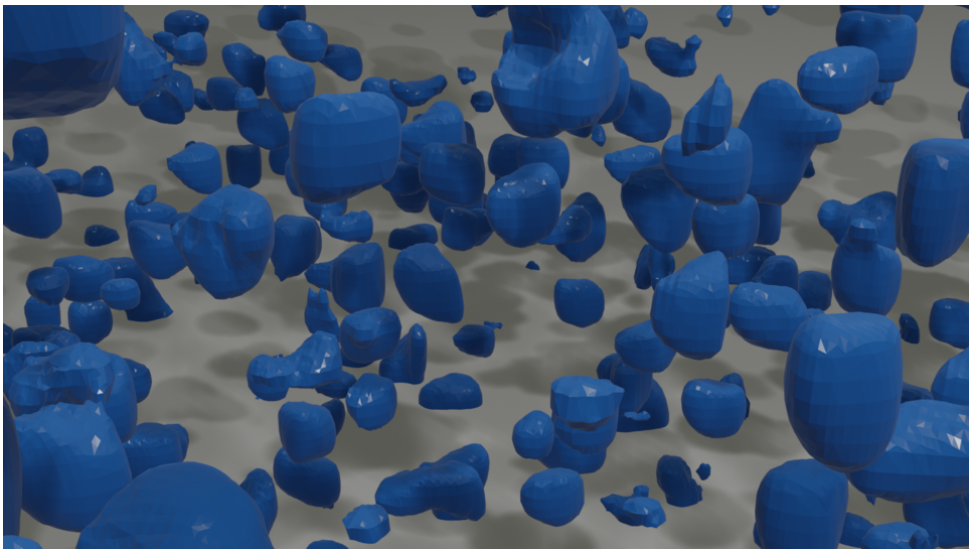

Figure 7.40: PD Dataset: Surface Reconstruction from 3D Predictions: Detail

An interesting difference between the segmentation provided by the 2D and 3D model is the disappearing of some *merging* artifacts in which two different objects are actually recognized a single extended one: in figure 7.41 we can see two objects that are already erroneously merged in GT and that UNET2D still reconstructs as

merged,(7.42), the 3D model instead recognizes the two objects as separate (figure 7.43 and 7.44), suggesting good generalization beyond unreliable GT data.

Figure 7.41: Merging Artifacts: GT Reconstructed Surface

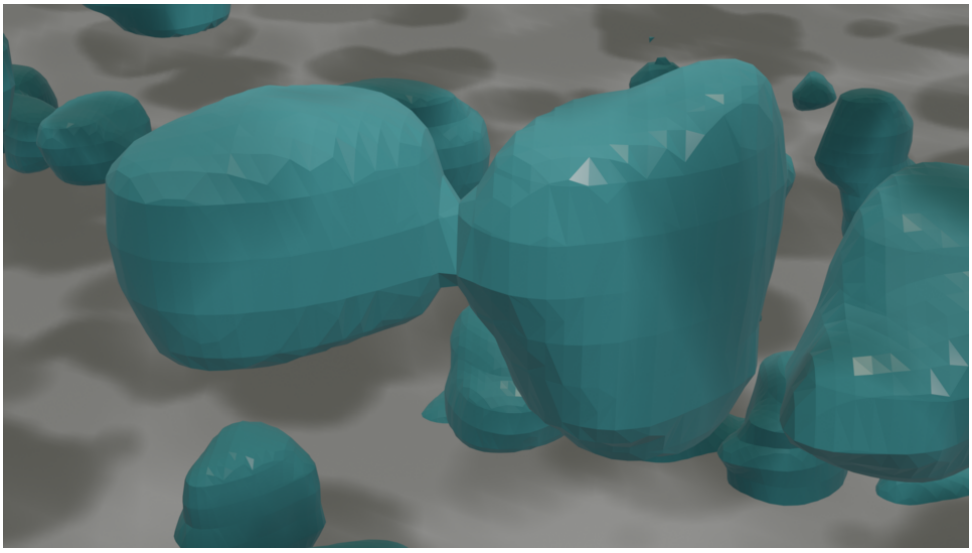

Figure 7.42: Merging Artifacts: 2D Reconstructed Surface

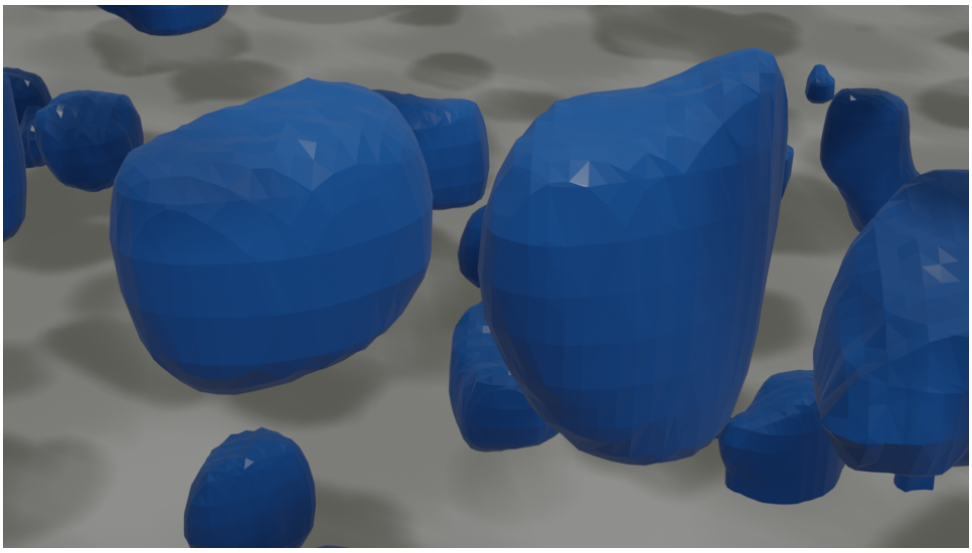

Figure 7.43: Merging Artifacts: 3D Reconstructed Surface

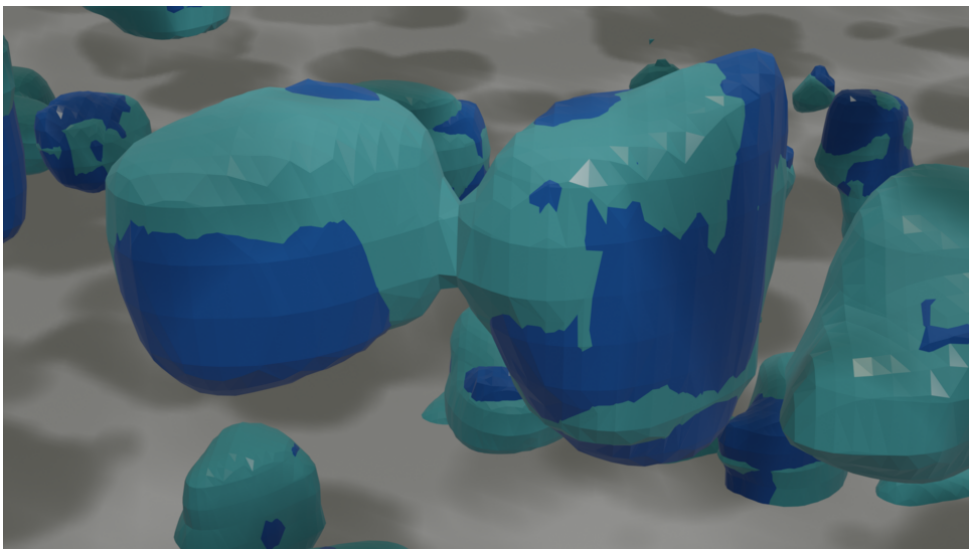

Figure 7.44: Merging Artifacts: Comparison between 2D and 3D Reconstructed Surfaces

Chapter 8

Conclusions

I think AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. If you have a large engine and a tiny amount of fuel, you won't make it to orbit. If you have a tiny engine and a ton of fuel, you can't even lift off. To build a rocket you need a huge engine and a lot of fuel.

Andrew Ng

8.1 On the Criticality of Data Availability

Large rockets need lots of fuel. The famous quote from Andrew Ng perfectly describes what is perhaps the biggest problems we encountered in training large models: the scarcity of extensively segmented data to train upon. If training on images requires large amounts of data, training on volumes geometrically worsens the problem. The main weapons against data scarcity (other than the obvious solution of *getting more data*) is *data augmentation*, but there are limits to the extent to which data can be effectively augmented, both from an analytical point of view (not every plausible data point can be reached via a simple transformation from another data point) and from a purely practical perspective: transforming data requires both computational power and efficient algorithms. If 2D data augmentation is a mainstream field of research with lots of interest, both from the academia and industry, and solutions for efficient data augmentation are continuously studied, implemented and deployed, the same can't be said about 3D data augmentation. Most of the times the generators

for data augmentation are manually implemented by the same groups who propose models and are not particularly computational efficient (that's surely the case of my own implemented data generators). This problematic directly translates to a tradeoff between computational times and effective dataset sizes. Results in this work were heavily limited by dataset size, computational resources and computational efficiency of my own data augmentation implementations, future work will surely benefit both from more powerful hardware and by more efficiency-aware algorithmic design.

8.2 Future Challenges: Multiview and Multichannel LSFM Segmentation

The proposed 3D CNN model cannot be considered a drop-in solution for the neuron segmentation problem even though a prospective shift on the data opens possibilities future research at LENS. LightSheet imaging with diSpim setups offer massive quantities of imaging information thanks to their multi-view and multi-channel capabilities: the possibility of fully exploiting such enormous imaging potential is yet to be explored and this work serves as an initial foundation for that (near) future effort.

On the multi-channel front, the proposed 3D model can be easily generalized to multi-channel operation using a multi-channel generalization of the 3D Convolution operation we used in this work: 3D Convolutions of a [W, H, L, C] sized input with a [k, k, d] sized filter is always possible as long as $L < d$

A major challenge is posed, though, by the presence of multiple views of the same volume. The diSpim lightsheet apparatus is capable of obtaining two orthogonal views of the subject, each with a resolution of roughly $1 \times 1 \times 5\mu m$. Analytic methods for combining the two different anisotropic resolution volumes into one single volume with isotropic resolution have been proposed by Wu et al [126] back in 2013. Although it might be an interesting option to train the models on isotropic resolutions, analytic volume reconstruction methods inevitably introduce artifacts. To solve the problem of multi-view data integration one could potentially follow two roads: one being feeding the network the isotropic resolution volumetric reconstructions and training on those, the other being experimenting with much more complex models with multiple paths that simultaneously process the corresponding volumes

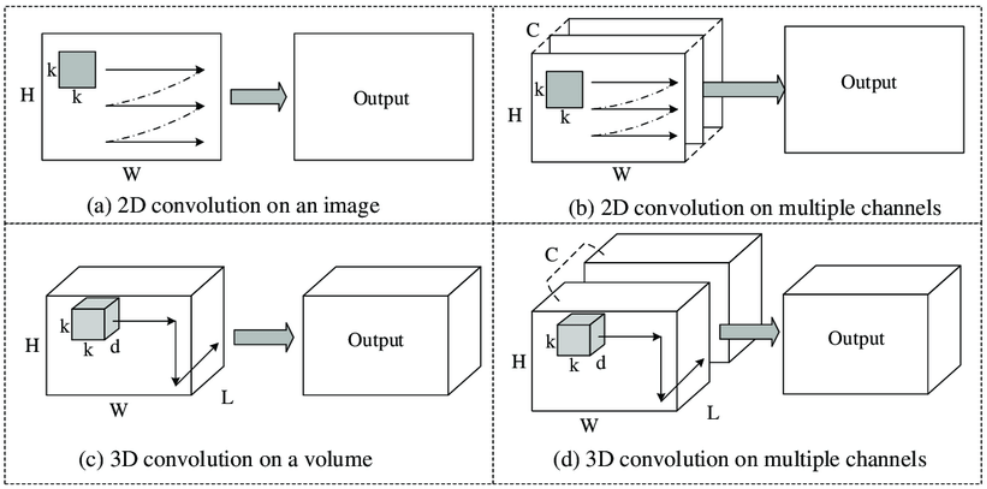

Figure 8.1: 3D Convolutions with Multiple Channels [74]

in both views and produce an unified isotropic response. The second one is surely an ambitious path but there are some results that point in that direction. In 2017 Kamnistas et al. produced a 3D model that integrated multi-scale views of the same subject with encouraging results [56], suggesting that models can actually benefit from multiview integration, and in 2018 Mylnarski et al. proposed the combination 3D CNNs with additional 2D features for segmentation improvement [89], proving the possibility of using additional features to standard 3D CNNs, even if they do not share the same volumetric format.

The road to multiview and multichannel CNN analysis in LSFM is a mostly unpaved one, with lots of questions yet to be answered: 3DCNNs can hopefully provide both the correct formulation and solution indications for the challenge.

Appendix A

Upsampling Artifacts

Specific checkerboard-like artifacts generated by *Transposed Convolution* layers were first observed in 2015 Gauthier in a work on Generative Adversarial Networks (GANs)[37]. The "mosaic-like" artifacts he observed were caused by the upsampling phases of the network and are the same that can appear in the outputs of a U-NET-like model: in figure A.3 we can see an example of that in the 2D model's outputs, as the object in the lower right clearly presents regular high-frequency noise.
The explanation for this kind of behavior is intuitive and can be traced to the way *transposed convolution* layers produce images: it is particularly clear by looking at figure A.2 that, when reconstructing the output image, it is likely to obtain uneven coverage of the output area if the size of the convolutional kernel is not divisible by the stride: some of the pixels are going to be exposed to more "passes" of the kernel than the other.

By extending the same reasoning to 2D Images (and, of course, to 3D ones), we can see that the effect is even more prominent, giving rise to checkerboard-style noise observed in A.1.

Figures A.2 and A.3 refer to a single transposed convolution layer but, when sequentially upscaling by chaining more transposed convolutions in the network graph (like in the U-NET and 3D-UNET cases), such effects can become even more noticeable and appear on multiple frequencies and scales, with early deep transposed convolutions creating low frequency noise and the last upsampling levels causing higher frequency patterns.

Figure A.1: Upsampling Artifacts in 2DCNN Outputs

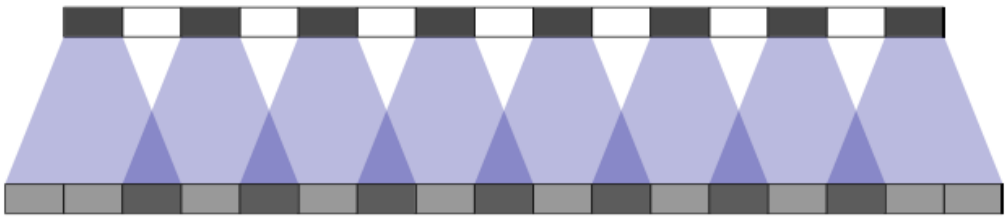

Figure A.2: Upsampling Artifacts: Uneven Coverage of Output Area [93]

The network, ideally, can eventually learn to overcome these effects by accordingly adjusting the weights in the transposed convolutions layers, as in figure

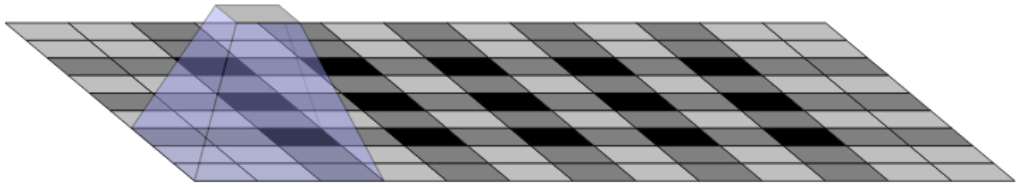

Figure A.3: Checkerboard Artifacts [93]

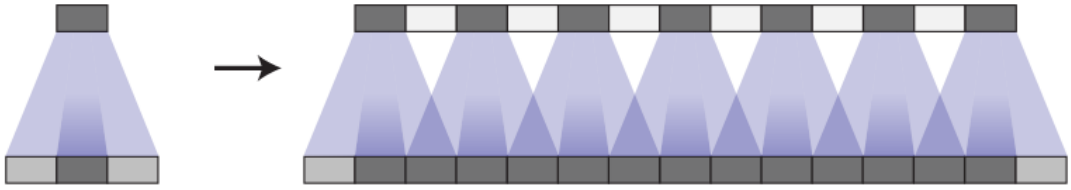

Figure A.4: Adjusting the Weights to Remove Artifacts [93]

it's to be noted, though, that this kind of adjustment reduces the effective representational capacity of the model by posing constraints on the relative importance of weights in the same kernel.

A possible strategy for reducing this problem could be using only strides that are divisible by the kernel size, but even then the model can sometimes learn kernels that present significant asymmetries in the weight distribution, causing similar effects. In figure A.5 we can observe that a kernel of size 4, when convolved using stride 2 (stride is divisible by kernel size), can cause similar artefacts if one weight is significantly higher than the others.

A solution that seems to eliminate at the base this kind of artifact is to completely decouple the upsampling and convolution phases by first resizing the input image using a bilinear or nearest neighbor upsampling and then applying a convolutional layer to the resized image.

It's a fun and interesting thing to notice that this problem has been known in

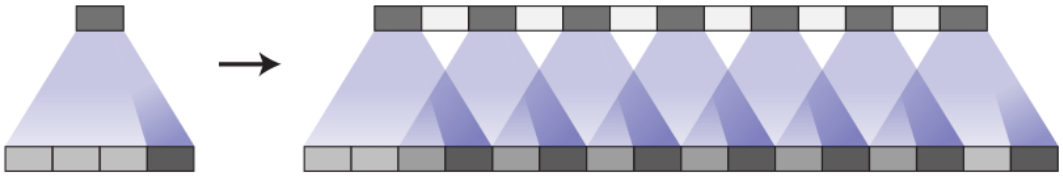

Figure A.5: Kernel Asymmetry Artifact [93]

the network visualization community for some years, although in a symmetrically different form. From a matrix multiplication perspective, the backpropagation pass of an ordinary convolution layer is actually equivalent to a transposed convolution in the forward pass, so the checkerboard artifacts we observe in the upsampled outputs are virtually the same that can be observed in the form of high frequency noise in model gradients: these artifacts are exactly the same that appear when trying to visualize a CNN model's gradient distribution in its various layers.

In conclusion: the existence of checkerboard artifacts is intrinsic to the nature of transposed convolutions and the problem can be alleviated using a kernel size divisible by the stride and can be completely avoided using standard convolutions after a deterministic upsampling layer.

Appendix B

Convolutions as Matrix Multiplications

Perhaps one of the most interesting properties of discrete convolutions is that they can be actually viewed as matrix multiplications of the inputs with a particularly constructed matrix. In the case of monodimensional and bidimensional convolutions the convolution matrix can be constructed using generalizations of what is known as a *Toeplitz matrix*: a *Toeplitz matrix* is a matrix in which values along the main diagonal and all the sub-diagonals are constant. An $N \times N$ Toeplitz matrix is entirely defined by a $(2N-1)$ long t_n sequence of elements ordered in such a way that

$$\{t_n | -(N-1) \leq n \leq (N-1)\} \tag{B.1}$$

Given the sequence $t_1 \dots t_{(2N-1}$ the Toeplitz matrix $T(m,n)$ can be constructed by simply placing the elements as

$$T(m,n) = T(m+1, n+1) \equiv t_{m-n} \tag{B.2}$$

In other words, you can obtain a Toeplitz matrix by putting the first half of the sequence in a descending order in the first column, starting from t_0 to $t_{(N-1)}$, shifting the sequence in the following column one element at the time so that the last

element disappears in $T((N-1), c)$ and a new element appears in position $T((0, c)$.

$$\begin{bmatrix} t_0 & t_{-1} & t_{-2} & \cdots & \cdots & \cdots & \cdots & t_{-(N-1)} \\ t_1 & t_0 & t_{-1} & t_{-2} & & & & \vdots \\ t_2 & t_1 & t_0 & t_{-1} & \ddots & & & \vdots \\ \vdots & t_2 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & t_{-2} & \vdots \\ \vdots & & & \ddots & t_1 & t_0 & t_{-1} & t_{-2} \\ \vdots & & & & t_2 & t_1 & t_0 & t_{-1} \\ t_{(N-1)} & \cdots & \cdots & \cdots & \cdots & t_2 & t_1 & t_0 \end{bmatrix} \tag{B.3}$$

We can extend this definition for series with positive-only indexes by replacing every negative-indexed element with 0: a Toeplitz 4×4 matrix for a $f[n] = \{1,2,3\}$ sequence shall be

$$T(f) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 0 & 3 & 2 & 1 \end{bmatrix} \tag{B.4}$$

B.1 1D Discrete Convolutions

Let's say we want to calculate the convolution of two monodimensional arrays:

$$\begin{aligned} h &= [h_0, h_1, h_2, h_3] \\ x &= [x_0, x_1, x_2] \end{aligned} \tag{B.5}$$

Which we can define as by the definition 1.27.

$$y_k = h * x = \sum_{i=-\infty}^{\infty} x_i h_{k-i} \quad k = 0,1,\ldots,(n+m-1=6) \tag{B.6}$$

Calculating one by one the terms of the convolution we can see a pattern:

$$\begin{aligned}
y_0 = &\ \textstyle\sum_{i=-\infty}^{\infty} x_i h_{-i} &&= x_0h_0 + 0 + 0 \\
y_1 = &\ \textstyle\sum_{i=-\infty}^{\infty} x_i h_{1-i} &&= x_0h_1 + x_1h_0 + 0 \\
y_2 = &\ \textstyle\sum_{i=-\infty}^{\infty} x_i h_{2-i} &&= x_0h_2 + x_1h_1 + x_2h_0 \\
y_3 = &\ \textstyle\sum_{i=-\infty}^{\infty} x_i h_{3-i} &&= x_0h_3 + x_1h_2 + x_2h_1 \\
y_4 = &\ \textstyle\sum_{i=-\infty}^{\infty} x_i h_{4-i} &&= x_1h_3 + x_2h_1 + 0 \\
y_5 = &\ \textstyle\sum_{i=-\infty}^{\infty} x_i h_{5-i} &&= x_2h_3 + 0 + 0
\end{aligned} \tag{B.7}$$

The convolution terms are exactly the same as what we would have obtained by multiplying the x array by a positive-term only 6×3 Toeplitz matrix built using $t_n = h_n$ as the generator sequence.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 \\ h_1 & h_0 & 0 \\ h_2 & h_1 & h_0 \\ h_3 & h_2 & h_1 \\ 0 & h_3 & h_2 \\ 0 & 0 & h_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{B.8}$$

Following the same reasoning we can write every convolution between two arbitrarily sized arrays

$$\begin{aligned} h &= [h_0, \ldots, h_m] \\ x &= [x_0, \ldots, x_n] \end{aligned} \tag{B.9}$$

as a multiplication between one of the arrays and the Toeplitz matrix obtained the other with sizes $m + n - 1 \times n$ or $m + n - 1 \times m$ in the two cases.[1]

[1] Because of the commutative property of convolution the order in which the operation is taken is not going to change the final result.

$$y = h * x = \begin{bmatrix} h_0 & 0 & \cdots & 0 & 0 \\ h_1 & h_0 & & \vdots & \vdots \\ h_2 & h_1 & \cdots & 0 & 0 \\ \vdots & h_2 & \cdots & h_0 & 0 \\ h_{m-1} & \vdots & \ddots & h_1 & h_0 \\ h_m & h_{m-1} & & \vdots & h_1 \\ 0 & h_m & \ddots & h_{m-2} & \vdots \\ 0 & 0 & \cdots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & & h_m & h_{m-1} \\ 0 & 0 & 0 & \cdots & h_m \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{B.10}$$

Transposed Convolutions can be calculated with the same ease, just using the transposition operator.

$$y^T = \begin{bmatrix} h_0 & h_1 & h_2 & \cdots & h_{m-1} & h_m \end{bmatrix} \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_n & 0 & 0 & 0 & \cdots & 0 \\ 0 & x_0 & x_1 & x_2 & \cdots & x_n & 0 & 0 & \cdots & 0 \\ 0 & 0 & x_0 & x_1 & x_2 & \dots & x_n & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & x_0 & \cdots & x_{n-2} & x_{n-1} & x_n & 0 \\ 0 & \cdots & 0 & 0 & 0 & x_0 & \cdots & x_{n-2} & x_{n-1} & x_n \end{bmatrix} \tag{B.11}$$

B.2 2D Discrete Convolutions

Calculating 2D convolution matrices is not conceptually different from calculating 1D ones, it just takes an additional tool in the Topelitz matrices framework, known as *Doubly Blocked Toeplitz Matrices*. Doubly Blocked Toeptliz Matrices (which for now on we're abbreviating in DBTMs for remembering the full name is a greater mental effort than actually understanding what they are) represent the natural extension of Toeplitz matrices to block matrices.

A block matrix A defined as

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{M1} & A_{M2} & \cdots & A_{MN} \end{bmatrix} \tag{B.12}$$

is a DBTM if

1. all of its individual blocks A_{ij} are Toeplitz Matrices
2. its structure with respect to its block is Toeplitz, i.e. $A_{i,j} = A_{i+1,j+1}$

Reformulating 2D discrete convolutions as matrix multiplications takes a bit more effort than the 1D case but ultimately applies the same concept. In the following paragraph we will show the general steps of 2D convolution conversion to a matrix multiplication problem without providing extensive justification of each step, as the purpose of this appendix is just to mention an equivalence - quite a crucial one when introducing transpose convolution - that most sources I encountered prefer to leave as implicit.

The first part of 2D discrete convolution conversion to matrix multiplication is figuring out the dimensions of the outputs: in the 1D case we had that convolving two arrays of m and n dimensions lead to an $m + n - 1$ output, by extending this reasoning to two dimensions it's trivial to conclude that the convolution of two 2D inputs sized respectively $m_1 \times n_1$ and $m_2 \times n_2$ is a $m_1 + m_2 - 1 \times n_1 + n_2 - 1$ output. This is the "full" size of a discrete convolution: "same" and "valid" convolutions can be obtained as submatrices of the "full" ouput.

If we want to convolve an 2×3 input I with a 2×2 filter F such that

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{B.13}$$

And let the filter be:

$$F = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \tag{B.14}$$

the resulting "full" output will be sized 3×4.

The next step is padding the filter matrix to have the same size as the output by adding zeros to the top and right sides of the filter, so that filter F becomes

$$F_{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 10 & 20 & 0 & 0 \\ 30 & 40 & 0 & 0 \end{bmatrix} \tag{B.15}$$

For each row of the padded filter a Toeplitz matrix with n_1 columns each is created with a resulting $m_1 + m_2 - 1 \times n_1$ size: in our example case we shall write three ($n_{\text{out}} = 3$) Toeplitz matrices from the rows of F_{padded}

$$\begin{aligned} F_0 &= \begin{bmatrix} 30 & 0 & 0 \\ 40 & 30 & 0 \\ 0 & 40 & 30 \\ 0 & 0 & 40 \end{bmatrix} \\ F_1 &= \begin{bmatrix} 10 & 0 & 0 \\ 20 & 10 & 0 \\ 0 & 20 & 10 \\ 0 & 0 & 20 \end{bmatrix} \\ F_2 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{B.16}$$

This sequence of Toeplitz matrices is now used to create a DBTM with n_1 block

columns:

$$C = \begin{bmatrix} F_0 & 0 \\ F_1 & F_0 \\ F_2 & F_1 \end{bmatrix}$$

$$= \begin{bmatrix} 30 & 0 & 0 & 0 & 0 & 0 \\ 40 & 30 & 0 & 0 & 0 & 0 \\ 0 & 40 & 30 & 0 & 0 & 0 \\ 0 & 0 & 40 & 0 & 0 & 0 \\ 10 & 0 & 0 & 30 & 0 & 0 \\ 20 & 10 & 0 & 40 & 30 & 0 \\ 0 & 20 & 10 & 0 & 40 & 30 \\ 0 & 0 & 20 & 0 & 0 & 40 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 20 & 10 & 0 \\ 0 & 0 & 0 & 0 & 20 & 10 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix} \tag{B.17}$$

The input matrix is then reshaped as a vector by sequentially writing its rows in inverse order, the so-obtained vector is to be multiplied with the matrix B.17 to have the final output the same vectorized format.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow I_{\text{vector}} = \begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \\ 2 \\ 3 \end{bmatrix} \tag{B.18}$$

$$O_{\text{vector}} = C \cdot I_{\text{vector}} = \begin{bmatrix} 30 & 0 & 0 & 0 & 0 & 0 \\ 40 & 30 & 0 & 0 & 0 & 0 \\ 0 & 40 & 30 & 0 & 0 & 0 \\ 0 & 0 & 40 & 0 & 0 & 0 \\ 10 & 0 & 0 & 30 & 0 & 0 \\ 20 & 10 & 0 & 40 & 30 & 0 \\ 0 & 20 & 10 & 0 & 40 & 30 \\ 0 & 0 & 20 & 0 & 0 & 40 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 20 & 10 & 0 \\ 0 & 0 & 0 & 0 & 20 & 10 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 120 \\ 310 \\ 380 \\ 240 \\ 70 \\ 230 \\ 330 \\ 240 \\ 10 \\ 40\ 70 \\ 60 \end{bmatrix} \tag{B.19}$$

$$O_{\text{vector}} = \begin{bmatrix} 120 \\ 310 \\ 380 \\ 240 \\ 70 \\ 230 \\ 330 \\ 240 \\ 10 \\ 40\ 70 \\ 60 \end{bmatrix} \rightarrow O = \begin{bmatrix} 10 & 40 & 70 & 60 \\ 70 & 230 & 330 & 240 \\ 120 & 310 & 380 & 240 \end{bmatrix} \tag{B.20}$$

The output matrix B.20 corresponds to the "full" discrete convolution: the "valid" convolution is obtained by just selecting the two central values $[230, 330]$. This is not the only algorithm for converting a 2D discrete convolution to a matrix multiplication and is particularly memory-consuming (matrix B.17 is most populated by zeroes) when confronted to others, but hopefully this small appendix is enough to pass the idea that, despite the unintuitive formulation, the convolution calculation problem is directly mappable to a series of easy matrix multiplications.

Bibliography

[1] Jayesh Bapu Ahire. *The XOR Problem in Neural Networks.* Dec. 2017. URL: https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b (visited on 06/19/2019).

[2] *Aliquis . Bioretics srl.* URL: https://www.bioretics.com/aliquis (visited on 08/19/2019).

[3] Stephen P. Amato et al. "Whole Brain Imaging with Serial Two-Photon Tomography". English. In: *Frontiers in Neuroanatomy* 10 (2016). ISSN: 1662-5129. DOI: 10.3389/fnana.2016.00031. URL: https://www.frontiersin.org/articles/10.3389/fnana.2016.00031/full (visited on 05/04/2019).

[4] Peter Atkins and Julio de Paula. *Atkins' Physical Chemistry.* Inglese. 9 edizione. Oxford ; New York: OUP Oxford, Nov. 2009. ISBN: 978-0-19-954337-3.

[5] Adriano Azaripour et al. "A survey of clearing techniques for 3D imaging of tissues with special reference to connective tissue". In: *Progress in Histochemistry and Cytochemistry* 51.2 (Aug. 2016), pp. 9–23. ISSN: 0079-6336. DOI: 10.1016/j.proghi.2016.04.001. URL: http://www.sciencedirect.com/science/article/pii/S0079633616300043 (visited on 05/09/2019).

[6] Daniele Bani, Tiziano Baroni, and Ennio Becchetti. *Istologia umana.* Italiano. Napoli: Idelson-Gnocchi, 2011. ISBN: 978-88-7947-541-9.

[7] Beniamino Barbieri. *A Short History of Fluorescence.* en. URL: https://fluorescence-foundation.org/lectures/madrid2010/lecture1.pdf.

[8] Klaus Becker et al. "Chemical Clearing and Dehydration of GFP Expressing Mouse Brains". In: *PLoS ONE* 7.3 (Mar. 2012). ISSN: 1932-6203. DOI: 10.1371/journal.pone.0033916. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3316521/ (visited on 05/09/2019).

[9] Floris van Beers. "Using Intersection over Union loss to improve Binary Image Segmentation". nl. bachelor. July 2018. URL: http://fse.studenttheses.ub.rug.nl/18139/ (visited on 07/03/2019).

[10] Bruce Blaus. *Types of Neuroglia.* Sept. 2013. URL: https://commons.wikimedia.org/wiki/File:Blausen_0870_TypesofNeuroglia.png (visited on 05/05/2019).

[11] Kendrick Boyd, Kevin H. Eng, and C. David Page. "Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals". en. In: *Advanced Information Systems Engineering*. Ed. by David Hutchison et al. Vol. 7908. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 451–466. ISBN: 978-3-642-38708-1 978-3-642-38709-8. DOI: 10.1007/978-3-642-40994-3_29. URL: http://link.springer.com/10.1007/978-3-642-40994-3_29 (visited on 09/05/2019).

[12] Per Brodal. *The Central Nervous System, Fourth Edition*. Inglese. 4 edizione. New York: OUP USA, Apr. 2010. ISBN: 978-0-19-538115-3.

[13] K. Brodmann. *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Open Library ID: OL25918683M. Leipzig: Barth, 1909.

[14] Renato Campanini et al. "A novel featureless approach to mass detection in digital mammograms based on support vector machines". en. In: *Physics in Medicine and Biology* 49.6 (Feb. 2004), pp. 961–975. ISSN: 0031-9155. DOI: 10.1088/0031-9155/49/6/007. URL: https://doi.org/10.1088%2F0031-9155%2F49%2F6%2F007 (visited on 09/05/2019).

[15] Renato Campanini et al. "A Support Vector Machine Classifier based on Recursive Feature Elimination for Microarray Data in Breast Cancer Characterization". In: Jan. 2002.

[16] Shan Carter et al. "Activation Atlas". In: *Distill* 4.3 (Mar. 2019), 10.23915/distill.00015. ISSN: 2476-0757. DOI: 10.23915/distill.00015. URL: https://distill.pub/2019/activation-atlas (visited on 06/13/2019).

[17] Filippo Castelli. *Generazione di Immagini Avversariali in Tensorflow*. original-date: 2018-04-25T15:13:40Z. Apr. 2019. URL: https://github.com/filippocastelli/adversarial_examples_tutorial-tensorflow (visited on 06/13/2019).

[18] C. Castilla et al. “Segmentation of actin-stained 3D fluorescent cells with filopodial protrusions using convolutional neural networks”. In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. Apr. 2018, pp. 413–417. DOI: 10.1109/ISBI.2018.8363605.

[19] Vladimir Cherkassky and Filip M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. en. John Wiley & Sons, Sept. 2007. ISBN: 978-0-470-14051-2.

[20] Guillaume Chevalier. *Smoothly Blend Image Patches*. original-date: 2017-08-25T20:26:36Z. Aug. 2019. URL: https://github.com/Vooban/Smoothly-Blend-Image-Patches (visited on 08/31/2019).

[21] Kwanghun Chung et al. “Structural and molecular interrogation of intact biological systems”. en. In: *Nature* 497.7449 (May 2013), pp. 332–337. ISSN: 1476-4687. DOI: 10.1038/nature12107. URL: https://www.nature.com/articles/nature12107 (visited on 05/10/2019).

[22] Davy Cielen, Arno Meysman, and Mohamed Ali. *Introducing Data Science: Big Data, Machine Learning, and more, using Python tools*. English. 1 edition. Shelter Island, NY: Manning Publications, May 2016. ISBN: 978-1-63343-003-7.

[23] Fiorenzo Conti. *Fisiologia medica: 1*. Italiano. 2 edizione. Milano: Edi. Ermes, 2005. ISBN: 978-88-7051-346-2.

[24] Irene Costantini et al. “A versatile clearing agent for multi-modal brain imaging”. en. In: *Scientific Reports* 5 (May 2015), p. 9808. ISSN: 2045-2322. DOI: 10.1038/srep09808. URL: https://www.nature.com/articles/srep09808 (visited on 05/07/2019).

[25] Gabriela Csurka, Diane Larlus, and Florent Perronnin. “What is a good evaluation measure for semantic segmentation?” en. In: *Procedings of the British Machine Vision Conference 2013*. Bristol: British Machine Vision Association, 2013, pp. 32.1–32.11. ISBN: 978-1-901725-49-0. DOI: 10.5244/C.27.32. URL: http://www.bmva.org/bmvc/2013/Papers/paper0032/index.html (visited on 07/19/2019).

[26] *Data Tables | Fluorescence Lifetime Standards | ISS*. URL: http://www.iss.com/resources/reference/data_tables/FL_LifetimeStandards.html (visited on 04/16/2019).

[27] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". en. In: *Proceedings of the 23rd international conference on Machine learning - ICML '06*. Pittsburgh, Pennsylvania: ACM Press, 2006, pp. 233–240. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143874. URL: http://portal.acm.org/citation.cfm?doid=1143844.1143874 (visited on 08/23/2019).

[28] *Deep neural networks: preventing overfitting*. July 2017. URL: https://www.jeremyjordan.me/deep-neural-networks-preventing-overfitting/ (visited on 07/03/2019).

[29] Alexey Dosovitskiy et al. "Learning to Generate Chairs, Tables and Cars with Convolutional Networks". In: *arXiv:1411.5928 [cs]* (Nov. 2014). arXiv: 1411.5928. URL: http://arxiv.org/abs/1411.5928 (visited on 07/28/2019).

[30] Chris Drummond and Robert C Holte. "What ROC Curves Can't Do (and Cost Curves Can)". en. In: (), p. 7.

[31] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv:1603.07285 [cs, stat]* (Mar. 2016). arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285 (visited on 07/28/2019).

[32] European Molecular Biology Laboratory (EMBL). *Digital Scanned Laser Light-sheet Microscopy - Structured Illumination (DSLM-SI)*. URL: https://www.youtube.com/watch?v=5xDN-4YLu-o (visited on 04/22/2019).

[33] *Fluorescence and Phosphorescence*. en. Oct. 2013. URL: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Spectroscopy/Electronic_Spectroscopy/Fluorescence_and_Phosphorescence (visited on 04/15/2019).

[34] *Fluorescence, Phosphorescence, Photoluminescence Differences*. en-GB. URL: https://www.edinst.com/blog/photoluminescence-differences/ (visited on 04/15/2019).

[35] *Fluorophores and Optical Filters for Fluorescence Microscopy*. en. URL: https://www.edmundoptics.com/resources/application-notes/optics/fluorophores-and-optical-filters-for-fluorescence-microscopy/ (visited on 04/16/2019).

[36] David Fortin et al. "Tractography in the study of the human brain: a neurosurgical perspective." In: *The Canadian journal of neurological sciences. Le journal canadien des sciences neurologiques* 39.6 (2012), pp. 747–756. DOI: 10.1017/S0317167100015560.

[37] Jon Gauthier. "Conditional generative adversarial nets for convolutional face generation". In: 2015.

[38] Marc Gellman and J. Rick Turner, eds. *Encyclopedia of Behavioral Medicine*. English. 2013 edition. New York: Springer, Dec. 2012. ISBN: 978-1-4419-1380-7.

[39] Antonino Paolo Di Giovanna et al. "Whole-Brain Vasculature Reconstruction at the Single Capillary Level". En. In: *Scientific Reports* 8.1 (Aug. 2018), p. 12573. ISSN: 2045-2322. DOI: 10.1038/s41598-018-30533-3. URL: https://www.nature.com/articles/s41598-018-30533-3 (visited on 05/10/2019).

[40] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Mar. 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html (visited on 07/31/2019).

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. en. Google-Books-ID: Np9SDQAAQBAJ. MIT Press, Nov. 2016. ISBN: 978-0-262-03561-3.

[42] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *arXiv:1412.6572 [cs, stat]* (Dec. 2014). arXiv: 1412.6572. URL: http://arxiv.org/abs/1412.6572 (visited on 06/13/2019).

[43] Mohammad Haft-Javaherian et al. “Deep convolutional neural networks for segmenting 3D in vivo multiphoton images of vasculature in Alzheimer disease mouse models”. en. In: *PLOS ONE* 14.3 (Mar. 2019), e0213539. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0213539. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0213539 (visited on 05/14/2019).

[44] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 08/25/2019).

[45] Kaiming He et al. “Identity Mappings in Deep Residual Networks”. In: *arXiv:1603.05027 [cs]* (Mar. 2016). arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027 (visited on 08/25/2019).

[46] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. en. Google-Books-ID: ddB4AgAAQBAJ. Psychology Press, Apr. 2005. ISBN: 978-1-135-63190-1.

[47] David Huang et al. “Optical Coherence Tomography”. In: *Science (New York, N.Y.)* 254.5035 (Nov. 1991), pp. 1178–1181. ISSN: 0036-8075. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4638169/ (visited on 05/10/2019).

[48] Jan Huisken and Didier Y. R. Stainier. “Selective plane illumination microscopy techniques in developmental biology”. In: *Development (Cambridge, England)* 136.12 (June 2009), pp. 1963–1975. ISSN: 0950-1991. DOI: 10.1242/dev.022426. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2685720/ (visited on 05/13/2019).

[49] *IBS - Photophysics of fluorescent proteins* -. URL: http://www.ibs.fr/research/research-groups/dynamics-and-kinetics-of-molecular-processes-group-m-weik/pixel/photophysics-of-fluorescent/ (visited on 04/13/2019).

[50] *Introduction: Fluorescence Microscopy - Soft Matter Physics Division - University of Leipzig*. URL: http://home.uni-leipzig.de/pwm/web/?section=introduction&page=fluorescence (visited on 04/13/2019).

[51] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]* (Feb. 2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (visited on 07/31/2019).

[52] Fabian Isensee et al. "Brain Tumor Segmentation and Radiomics Survival Prediction: Contribution to the BRATS 2017 Challenge". In: *arXiv:1802.10508 [cs]* (Feb. 2018). arXiv: 1802.10508. URL: http://arxiv.org/abs/1802.10508 (visited on 07/29/2019).

[53] A. Jablonski. "Efficiency of Anti-Stokes Fluorescence in Dyes". En. In: *Nature* 131.3319 (June 1933), p. 839. ISSN: 1476-4687. DOI: 10.1038/131839b0. URL: https://www.nature.com/articles/131839b0 (visited on 04/13/2019).

[54] *Jablonski diagram*. en. Oct. 2013. URL: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Spectroscopy/Electronic_Spectroscopy/Jablonski_diagram (visited on 04/13/2019).

[55] Justin Johnson. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/ (visited on 07/28/2019).

[56] Konstantinos Kamnitsas et al. "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation". In: *Medical Image Analysis* 36 (Feb. 2017), pp. 61–78. ISSN: 1361-8415. DOI: 10.1016/j.media.2016.10.004. URL: http://www.sciencedirect.com/science/article/pii/S1361841516301839 (visited on 05/13/2019).

[57] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell. *Principi di neuroscienze. Volume unico*. Italiano. Trans. by V. Perri and G. Spidalieri. 4 edizione. Rozzano (MI): CEA, Nov. 2014. ISBN: 978-88-08-18445-0.

[58] Philipp J. Keller et al. "Digital scanned laser light-sheet fluorescence microscopy (DSLM) of zebrafish and Drosophila embryonic development". eng. In: *Cold Spring Harbor Protocols* 2011.10 (Oct. 2011), pp. 1235–1243. ISSN: 1559-6095. DOI: 10.1101/pdb.prot065839.

[59] Meenakshi Khosla et al. “3D Convolutional Neural Networks for Classification of Functional Connectomes”. In: *arXiv:1806.04209 [cs, stat]* (June 2018). arXiv: 1806.04209. URL: http://arxiv.org/abs/1806.04209 (visited on 05/13/2019).

[60] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 08/03/2019).

[61] Martina Kottas, Oliver Kuss, and Antonia Zapf. “A modified Wald interval for the area under the ROC curve (AUC) in diagnostic case-control studies”. en. In: *BMC Medical Research Methodology* 14.1 (Dec. 2014), p. 26. ISSN: 1471-2288. DOI: 10.1186/1471-2288-14-26. URL: http://bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-14-26 (visited on 09/05/2019).

[62] Mateusz Koziński et al. “Learning to Segment 3D Linear Structures Using Only 2D Annotations”. en. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*. Ed. by Alejandro F. Frangi et al. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 283–291. ISBN: 978-3-030-00934-2.

[63] Uros Krzic et al. “Multiview light-sheet microscope for rapid in toto imaging”. eng. In: *Nature Methods* 9.7 (June 2012), pp. 730–733. ISSN: 1548-7105. DOI: 10.1038/nmeth.2064.

[64] Ulrich Kubitscheck, ed. *Fluorescence Microscopy: From Principles to Biological Applications*. English. 2 edition. Weinheim: Wiley-Blackwell, June 2017. ISBN: 978-3-527-33837-5.

[65] Andrey Kurenov. *A ’Brief’ History of Neural Nets and Deep Learning*. en. URL: /writing/ai/a-brief-history-of-neural-nets-and-deep-learning/ (visited on 06/19/2019).

[66] *LAIRA - "Laira is an AI-based Research Assistant" - Bioretics srl*. URL: https://laira.bioretics.com/ (visited on 08/19/2019).

[67] Joseph R. Lakowicz. *Principles of Fluorescence Spectroscopy*. en. 3rd ed. Springer US, 2006. ISBN: 978-0-387-31278-1. URL: https://www.springer.com/it/book/9780387312781 (visited on 04/13/2019).

[68] Nieradzik Lars. *Losses for Image Segmentation*. en. Sept. 2018. URL: https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/ (visited on 06/24/2019).

[69] Yann A. LeCun et al. "Efficient BackProp". en. In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3 (visited on 08/07/2019).

[70] Thomas Lewiner et al. "Efficient Implementation of Marching Cubes' Cases with Topological Guarantees". en. In: *Journal of Graphics Tools* 8.2 (Jan. 2003), pp. 1–15. ISSN: 1086-7651. DOI: 10.1080/10867651.2003.10487582. URL: http://www.tandfonline.com/doi/abs/10.1080/10867651.2003.10487582 (visited on 08/19/2019).

[71] R. Li et al. "Deep Learning Segmentation of Optical Microscopy Images Improves 3-D Neuron Reconstruction". In: *IEEE Transactions on Medical Imaging* 36.7 (July 2017), pp. 1533–1541. ISSN: 0278-0062. DOI: 10.1109/TMI.2017.2679713.

[72] Chunfeng Lian et al. "Multi-channel multi-scale fully convolutional network for 3D perivascular spaces segmentation in 7T MR images". In: *Medical Image Analysis* 46 (May 2018), pp. 106–117. ISSN: 1361-8415. DOI: 10.1016/j.media.2018.02.009. URL: http://www.sciencedirect.com/science/article/pii/S1361841518300409 (visited on 06/28/2019).

[73] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: *arXiv:1312.4400 [cs]* (Dec. 2013). arXiv: 1312.4400. URL: http://arxiv.org/abs/1312.4400 (visited on 08/17/2019).

[74] Zhiqiang Liu et al. "A Uniform Architecture Design for Accelerating 2D and 3D CNNs on FPGAs". In: *Electronics* 8 (Jan. 2019), p. 65. DOI: 10.3390/electronics8010065.

[75] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". en. In: (), p. 10.

[76] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169. ISBN: 0-89791-227-6. DOI: 10.1145/37401.37422. URL: http://doi.acm.org/10.1145/37401.37422.

[77] Aurelien Lucchi, Yunpeng Li, and Pascal Fua. "Learning for Structured Prediction Using Approximate Subgradient Descent with Working Sets". en. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. Portland, OR, USA: IEEE, June 2013, pp. 1987–1994. ISBN: 978-0-7695-4989-7. DOI: 10.1109/CVPR.2013.259. URL: http://ieeexplore.ieee.org/document/6619103/ (visited on 08/05/2019).

[78] Douglas Magde, Gail E. Rojas, and Paul G. Seybold. "Solvent Dependence of the Fluorescence Lifetimes of Xanthene Dyes". en. In: *Photochemistry and Photobiology* 70.5 (1999), pp. 737–744. ISSN: 1751-1097. DOI: 10.1111/j.1751-1097.1999.tb08277.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-1097.1999.tb08277.x (visited on 04/16/2019).

[79] Caroline Magnain et al. "Optical coherence tomography visualizes neurons in human entorhinal cortex". In: *Neurophotonics* 2.1 (Jan. 2015). ISSN: 2329-423X. DOI: 10.1117/1.NPh.2.1.015004. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4346095/ (visited on 05/04/2019).

[80] G. Mazzamuto et al. "Automatic Segmentation of Neurons in 3D Samples of Human Brain Cortex". en. In: *Applications of Evolutionary Computation*. Ed. by Kevin Sim and Paul Kaufmann. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 78–85. ISBN: 978-3-319-77538-8.

[81] Giacomo Mazzamuto et al. *Software Tools for Efficient Processing of High-Resolution 3D Images of Macroscopic Brain Samples*. URL: https://www.osapublishing.org/abstract.cfm?uri=Microscopy-2018-JTh3A.64 (visited on 08/19/2019).

[82] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". en. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259 (visited on 06/18/2019).

[83] Jing Men et al. "Optical Coherence Tomography for Brain Imaging and Developmental Biology". In: *IEEE journal of selected topics in quantum electronics : a publication of the IEEE Lasers and Electro-optics Society* 22.4 (2016). ISSN: 1077-260X. DOI: 10.1109/JSTQE.2015.2513667. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5049888/ (visited on 05/10/2019).

[84] A. Mikołajczyk and M. Grochowski. "Data augmentation for improving deep learning in image classification problem". In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. May 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[85] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *arXiv:1606.04797 [cs]* (June 2016). arXiv: 1606.04797. URL: http://arxiv.org/abs/1606.04797 (visited on 07/03/2019).

[86] Marvin Lee Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. en. Google-Books-ID: Ow1OAQAAIAAJ. Mit Press, 1972. ISBN: 978-0-262-63022-1.

[87] Tom M. Mitchell. *Machine Learning*. English. 1 edition. New York: McGraw-Hill Education, Mar. 1997. ISBN: 978-0-07-042807-2.

[88] *Mitochondria Detection in EM Stacks – CVLAB*. en-GB. URL: https://cvlab.epfl.ch/research/page-90578-en-html/research-medical-em-mitochondria-index-php/ (visited on 06/28/2019).

[89] Pawel Mlynarski et al. "3D Convolutional Neural Networks for Tumor Segmentation using Long-range 2D Context". In: *arXiv:1807.08599 [cs]* (July 2018). arXiv: 1807.08599. URL: http://arxiv.org/abs/1807.08599 (visited on 05/13/2019).

[90] Evan Murray et al. "Simple, scalable proteomic imaging for high-dimensional profiling of intact systems". In: *Cell* 163.6 (Dec. 2015), pp. 1500–1514. ISSN: 0092-8674. DOI: 10.1016/j.cell.2015.11.025. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5275966/ (visited on 05/13/2019).

[91] Mark Muyskens and Ed Vitz. "The Fluorescence of Lignum nephriticum: A Flash Back to the Past and a Simple Demonstration of Natural Substance Fluorescence". In: *Journal of Chemical Education* 83.5 (May 2006), p. 765. ISSN: 0021-9584. DOI: 10.1021/ed083p765. URL: https://doi.org/10.1021/ed083p765 (visited on 04/16/2019).

[92] M. Caroline Müllenbroich et al. "High-Fidelity Imaging in Brain-Wide Structural Studies Using Light-Sheet Microscopy". eng. In: *eNeuro* 5.6 (Dec. 2018). ISSN: 2373-2822. DOI: 10.1523/ENEURO.0124-18.2018.

[93] Augustus Odena, Vincent Dumuolin, and Chris Olah. *Deconvolution and Checkerboard Artifacts*. URL: https://distill.pub/2016/deconv-checkerboard/ (visited on 08/07/2019).

[94] Omar E. Olarte et al. "Light-sheet microscopy: a tutorial". EN. In: *Advances in Optics and Photonics* 10.1 (Mar. 2018), pp. 111–179. ISSN: 1943-8206. DOI: 10.1364/AOP.10.000111. URL: https://www.osapublishing.org/aop/abstract.cfm?uri=aop-10-1-111 (visited on 04/23/2019).

[95] Diana Porro-Muñoz et al. "Tractome: a visual data mining tool for brain connectivity analysis". en. In: *Data Mining and Knowledge Discovery* 29.5 (Sept. 2015), pp. 1258–1279. ISSN: 1573-756X. DOI: 10.1007/s10618-015-0408-z. URL: https://doi.org/10.1007/s10618-015-0408-z (visited on 05/10/2019).

[96] Emmanuel G. Reynaud et al. "Guide to light-sheet microscopy for adventurous biologists". en. In: *Nature Methods* 12 (Dec. 2014), pp. 30–34. ISSN: 1548-7105. DOI: 10.1038/nmeth.3222. URL: https://www.nature.com/articles/nmeth.3222 (visited on 04/24/2019).

[97] Hamid Rezatofighi et al. "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression". In: *arXiv:1902.09630 [cs]* (Feb. 2019). arXiv: 1902.09630. URL: http://arxiv.org/abs/1902.09630 (visited on 07/03/2019).

[98] Matteo Roffilli. "Advanced Machine Learning Techniques for Digital Mammography". PhD Thesis. Tech. Rep. UBLCS-2006-12, University of Bologna, Mar. 2006. URL: http://www.cs.unibo.it/people/phd-students/roffilli/.

[99] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv:1505.04597 [cs]* (May 2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597 (visited on 05/13/2019).

[100] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain". eng. In: *Psychological Review* 65.6 (Nov. 1958), pp. 386–408. ISSN: 0033-295X.

[101] Ali Salehi. *Step by step explanation of 2D convolution implemented as matrix multiplication using toeplitz matrices: alisaaalehi/convolution_as_multiplication*. original-date: 2018-08-15T04:46:54Z. July 2019. URL: https://github.com/alisaaalehi/convolution_as_multiplication (visited on 08/14/2019).

[102] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (July 1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[103] Olaf Selchow and Jan Huisken. "Lichtblattmikroskopie: Das Beleuchtungskonzept revolutioniert die 3D-Analyse lebender Proben". de. In: (), p. 4.

[104] Pierre Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *arXiv:1312.6229 [cs]* (Dec. 2013). arXiv: 1312.6229. URL: http://arxiv.org/abs/1312.6229 (visited on 08/13/2019).

[105] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". en. In: *Journal of Big Data* 6.1 (Dec. 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0 (visited on 08/03/2019).

[106] L. Silvestri et al. "Confocal light sheet microscopy: micron-scale neuroanatomy of the entire mouse brain". EN. In: *Optics Express* 20.18 (Aug. 2012), pp. 20582–20598. ISSN: 1094-4087. DOI: 10.1364/OE.20.020582. URL: https://www.osapublishing.org/oe/abstract.cfm?uri=oe-20-18-20582 (visited on 04/22/2019).

[107] Ludovico Silvestri et al. "Clearing of fixed tissue: a review from a microscopist's perspective". In: *Journal of Biomedical Optics* 21.8 (Mar. 2016), p. 081205. ISSN: 1083-3668, 1560-2281. DOI: 10.1117/1.JBO.21.8.081205. URL: https://www.spiedigitallibrary.org/journals/Journal-of-Biomedical-Optics/volume-21/issue-8/081205/Clearing-of-fixed-tissue--a-review-from-a-microscopists/10.1117/1.JBO.21.8.081205.short (visited on 05/09/2019).

[108] Ludovico Silvestri et al. "Towards a Full Volumetric Atlas of Cell-specific Neuronal Spatial Organization in the Entire Mouse Brain". EN. In: *Biophotonics Congress: Biomedical Optics Congress 2018 (Microscopy/Translational/Brain/OTS) (2018), paper JTu3A.62*. Optical Society of America, Apr. 2018, JTu3A.62. DOI: 10.1364/TRANSLATIONAL.2018.JTu3A.62. URL: https://www.osapublishing.org/abstract.cfm?uri=BRAIN-2018-JTu3A.62 (visited on 06/29/2019).

[109] P. Y. Simard, D. Steinkraus, and J. C. Platt. "Best practices for convolutional neural networks applied to visual document analysis". In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* Aug. 2003, pp. 958–963. DOI: 10.1109/ICDAR.2003.1227801.

[110] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". en. In: *arXiv:1409.1556 [cs]* (Sept. 2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 08/25/2019).

[111] Larry Squire et al. *Fundamental Neuroscience*. en. Google-Books-ID: QGzJFu_NyzcC. Academic Press, Dec. 2012. ISBN: 978-0-12-385871-9.

[112] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 07/06/2019).

[113] *Staining methods: Staining of nerve tissue*. URL: https://pathologycenter.jp/crrinpa/crrinpa10.html (visited on 05/06/2019).

[114] G. Stokes. *On the Change of Refrangibility of Light*. eng. Royal Society of London, Jan. 1852. URL: http://archive.org/details/philtrans03348300 (visited on 04/13/2019).

[115] Carole H. Sudre et al. "Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations". In: *arXiv:1707.03237 [cs]* 10553 (2017). arXiv: 1707.03237, pp. 240–248. DOI: 10.1007/978-3-319-67558-9_28. URL: http://arxiv.org/abs/1707.03237 (visited on 07/03/2019).

[116] Karel Svoboda and Ryohei Yasuda. "Principles of Two-Photon Excitation Microscopy and Its Applications to Neuroscience". In: *Neuron* 50.6 (June 2006), pp. 823–839. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2006.05.019. URL: http://www.sciencedirect.com/science/article/pii/S0896627306004119 (visited on 04/16/2019).

[117] Abdel Aziz Taha and Allan Hanbury. "Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool". In: *BMC Medical Imaging* 15 (Aug. 2015). ISSN: 1471-2342. DOI: 10.1186/s12880-015-0068-x. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4533825/ (visited on 08/22/2019).

[118] Giles Tetteh et al. "DeepVesselNet: Vessel Segmentation, Centerline Prediction, and Bifurcation Detection in 3-D Angiographic Volumes". In: *arXiv:1803.09340 [cs]* (Mar. 2018). arXiv: 1803.09340. URL: http://arxiv.org/abs/1803.09340 (visited on 06/28/2019).

[119] Yuta Tokuoka et al. "Convolutional Neural Network-Based Instance Segmentation Algorithm to Acquire Quantitative Criteria of Early Mouse Development". en. In: *bioRxiv* (June 2018), p. 324186. DOI: 10.1101/324186. URL: https://www.biorxiv.org/content/10.1101/324186v3 (visited on 06/21/2019).

[120] *Types of neurons*. en. Nov. 2017. URL: https://qbi.uq.edu.au/brain/brain-anatomy/types-neurons (visited on 05/03/2019).

[121] Juan P. Vigueras-Guillén et al. "Fully convolutional architecture vs sliding-window CNN for corneal endothelium cell segmentation". In: *BMC Biomedical Engineering* 1.1 (Jan. 2019), p. 4. ISSN: 2524-4426. DOI: 10.1186/s42490-

019-0003-2. URL: https://doi.org/10.1186/s42490-019-0003-2 (visited on 07/28/2019).

[122] Chengjia Wang et al. "A two-stage 3D Unet framework for multi-class segmentation on full resolution image". In: *arXiv:1804.04341 [cs]* (Apr. 2018). arXiv: 1804.04341. URL: http://arxiv.org/abs/1804.04341 (visited on 05/13/2019).

[123] Hui Wang et al. "Reconstructing micrometer-scale fiber pathways in the brain: multi-contrast optical coherence tomography based tractography". In: *NeuroImage* 58.4 (Oct. 2011), pp. 984–992. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2011.07.005. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3178460/ (visited on 05/10/2019).

[124] Michael Weber, Michaela Mickoleit, and Jan Huisken. "Light sheet microscopy". eng. In: *Methods in Cell Biology* 123 (2014), pp. 193–215. ISSN: 0091-679X. DOI: 10.1016/B978-0-12-420138-5.00011-2.

[125] Paul John Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. en. Google-Books-ID: z81XmgEACAAJ. Harvard University, 1975.

[126] Yicong Wu et al. "Spatially isotropic four-dimensional imaging with dual-view plane illumination microscopy". In: *Nature biotechnology* 31.11 (Nov. 2013), pp. 1032–1038. ISSN: 1087-0156. DOI: 10.1038/nbt.2713. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4105320/ (visited on 08/07/2019).

[127] Chris Xu and Watt W. Webb. "Measurement of two-photon excitation cross sections of molecular fluorophores with data from 690 to 1050 nm". en. In: *Journal of the Optical Society of America B* 13.3 (Mar. 1996), p. 481. ISSN: 0740-3224, 1520-8540. DOI: 10.1364/JOSAB.13.000481. URL: https://www.osapublishing.org/abstract.cfm?URI=josab-13-3-481 (visited on 09/04/2019).

[128] Jiahui Yu et al. "UnitBox: An Advanced Object Detection Network". In: *Proceedings of the 2016 ACM on Multimedia Conference - MM '16* (2016). arXiv: 1608.01471, pp. 516–520. DOI: 10.1145/2964284.2967274. URL: http://arxiv.org/abs/1608.01471 (visited on 07/03/2019).

[129] Lequan Yu et al. “Volumetric ConvNets with Mixed Residual Connections for Automated Prostate Segmentation from 3D MR Images”. en. In: (), p. 7.

[130] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. en. In: *arXiv:1605.07146 [cs]* (May 2016). arXiv: 1605.07146. URL: http://arxiv.org/abs/1605.07146 (visited on 08/25/2019).

[131] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. “Road Extraction by Deep Residual U-Net”. en. In: *IEEE Geoscience and Remote Sensing Letters* 15.5 (May 2018). arXiv: 1711.10684, pp. 749–753. ISSN: 1545-598X, 1558-0571. DOI: 10.1109/LGRS.2018.2802944. URL: http://arxiv.org/abs/1711.10684 (visited on 08/25/2019).

[132] Z. Zhong et al. “3D fully convolutional networks for co-segmentation of tumors on PET-CT images”. In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. Apr. 2018, pp. 228–231. DOI: 10.1109/ISBI.2018.8363561.

[133] Özgün Çiçek et al. “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation”. In: *arXiv:1606.06650 [cs]* (June 2016). arXiv: 1606.06650. URL: http://arxiv.org/abs/1606.06650 (visited on 05/13/2019).